МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ІНФОРМАТИКА, ОБЧИСЛЮВАЛЬНА ТЕХНІКА, ПРОГРАМУВАННЯ ТА ЧИСЛОВІ МЕТОДИ

КОМП'ЮТЕРНИЙ ПРАКТИКУМ

(Частина 1)

*Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для студентів,
які навчаються за спеціальністю 132 «Матеріалознавство»*

Київ
КПІ ім. Ігоря Сікорського
2019

Інформатика, обчислювальна техніка, програмування та числові методи: Комп'ютерний практикум (Частина 1) [Електронний ресурс] : навч. посіб. для студ. спеціальності 132 «Матеріалознавство» / КПІ ім. Ігоря Сікорського; уклад.: О. В. Степанов, Є. Г. Биба, Т. О. Соловйова. – Електронні текстові дані (1 файл: 4,18 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2019. – 135 с.

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського (протокол № 2 від 31.10.2019 р.) за поданням Вченої ради Інженерно-фізичного факультету (протокол № 09/19 від 30.09.2019 р.)

Електронне мережне навчальне видання

ІНФОРМАТИКА, ОБЧИСЛЮВАЛЬНА ТЕХНІКА, ПРОГРАМУВАННЯ ТА ЧИСЛОВІ МЕТОДИ
КОМП'ЮТЕРНИЙ ПРАКТИКУМ
(Частина 1)

Укладачі: *Степанов Олег Васильович*, канд. техн. наук, доц.
Биба Євген Георгійович, канд. техн. наук
Соловйова Тетяна Олександрівна, канд. техн. наук

Відповідальний редактор *Богомол Ю. І.,* кандидат технічних наук, доцент

Рецензент *Доній О. М.,* кандидат технічних наук, доцент

Дисципліна "Інформатика, обчислювальна техніка, програмування та числові методи" є вступним курсом з програмування, що дає уявлення про базові поняття структурного програмування (даних, операціях, змінних, розгалуженнях в програмі, циклах і функціях). Вибір мови *Python* обумовлений низкою переваг щодо інших мов для початківців: ясність коду, швидкість реалізації.

Структура комп'ютерного практикуму складена таким чином, що використовуючи його, студент має можливість ознайомитись з мовою програмування *Python*.

До навчального посібника увійшли 10 комп'ютерних практикумів, кожен з яких складається з двох основних розділів: основні теоретичні відомості та завдання на практикум.

В експериментальній частині, окрім порядку виконання практикуму та вимог до оформлення звіту, наведено правила безпеки під час виконання роботи. Кожен практикум закінчується контрольними питаннями, які дозволяють студенту провести самоаналіз готовності до роботи.

© КПІ ім. Ігоря Сікорського, 2019

ЗМІСТ

ВСТУП

Сьогодні практично неможливо уявити будь-яку сферу де б не використовувалося програмування. Поняття комп'ютерного програмування носять логічний та математичний характер. У теорії, комп'ютерні програми можуть бути розроблені без використання комп'ютера. Програмісти можуть оцінити життєздатність програми її правильність та ефективність, використовуючи абстрактні символи, які відповідають особливостям реальних мов програмування, але не відображаються в жодній реальній мові програмування. Тому й виникає питання: "Яке ж програмування більш доступне та зрозуміле для користувача – початківця, функціональне чи імперативне?". Їх порівняння носить досить контрастний характер, незважаючи на те, що функціональне програмування також не є об'єктно-орієнтованим, деякі його мови є достатньо успішним симбіозом. Однією з найважливіших концепцій функціонального програмування є "чиста функція", яка завжди повертає один і той же результат, з огляду на одні й ті ж аргументи, що більш відповідає значенню "функції" в математиці, ніж у імперативному програмуванні.

Серед великої кількості мов програмування все більше користувачів віддають перевагу зрозумілому та нескладному коду. Поєднуючи повноцінність та простоту у використанні, наразі мова *Python* стала найбільш розповсюдженою, високорівневою та сучасною мовою програмування. Завдяки універсальності *Python* може застосовуватися практично до будь-якого завдання програмування, дозволяючи швидко розгалужувати та відлагоджувати, як окремі блоки так і програму вцілому. *Python*, безумовно, не є "чистою функціональною мовою програмування"; побічні ефекти широко розповсюджені в більшості програм *Python*. Тобто, змінні часто відновлюються, масиви даних часто змінюють вміст, а введення-виведення вільно перетинається з обчисленнями. В цілому ж це навіть не "функціональна мова програмування". Тим не менш, *Python* – це багатопарадигмальна мова, що робить функціональне програмування зручним, якщо це потрібно, і легко змішується з іншими стилями програмування.

ТЕХНІКА БЕЗПЕКИ ПРИ РОБОТІ З КОМП'ЮТЕРОМ. ПРАВИЛА ПОВЕДІНКИ В КОМП'ЮТЕРНОМУ КЛАСІ

Розпочинаючи працювати на ПК, необхідно пам`ятати, що це дуже складна апаратура, яка потребує акуратного й обережного ставлення до неї, високої самодисципліни на всіх етапах її експлуатації. Напруга живлення ПК (220 В) є небезпечною для життя людини. Тому, незважаючи на те, що в конструкції комп`ютера передбачена достатня ізоляція від струмопровідних ділянок, необхідно знати та чітко виконувати ряд правил техніки безпеки.

ЗАБОРОНЯЄТЬСЯ:

- знаходитися в класі у верхньому одязі;
- класти одяг і речі на столи;
- працювати за комп`ютером у вологому одязі та вологими руками;
- торкатись екрана, тильного боку дисплея, проводів живлення, заземлення, з`єднувальних кабелів;
- пересувати комп'ютери і монітори, знімати кришку корпусу системного блоку;
- вмикати/вимикати комп'ютер без дозволу, від'єднувати і під’єднувати будь-які пристрої комп'ютера, порушувати порядок увімкнення й вимикання апаратних блоків;
- самостійно намагатися усунути будь-які неполадки в роботі комп'ютера, незалежно від того, коли і з чиєї вини вони сталися;
- перекривати вентиляційні отвори на системному блоці та моніторі;
- класти книги, зошити та інші речі на клавіатуру, монітор і системний блок;
- видаляти і переміщати чужі файли, приносити і запускати комп'ютерні ігри.
- приносити, вживати їжу та напої в комп'ютерному в класі;

Перед початком роботи на комп`ютері необхідно отримати дозвіл на роботу у викладача.

Під час роботи на комп`ютері **НЕОБХІДНО**:

– дотримуватись тиші і порядку;
– працювати на клавіатурі чистими сухими руками, не натискуючи на клавіші без потреби чи навмання;
– коректно завершувати роботу з програмним забезпеченням.

У разі появи запаху горілого, самовільного вимикання апаратури, незвичних звуків треба негайно повідомити про це викладача і виконувати його вказівки. Не можна працювати на комп`ютері при недостатньому освітленні, високому рівні шуму тощо. Необхідно, щоб очі користувача знаходилися на відстані 60-70 см від екрана, а безперервна робота за комп`ютером тривала не більше 45-90 хв.

Інструктаж щодо виконання комп’ютерних практикумів проводиться на першому занятті і фіксується в журналі техніки безпеки.

ОФОРМЛЕННЯ ЗВІТУ ТА ПОРЯДОК ЙОГО ПОДАННЯ

Для допуску до виконання кожного практикуму студент має підготувати протокол в який заносяться принципові положення теоретичних відомостей та власні ескізи програмного методу, що реалізують тему відповідного практикуму. Протокол готується рукописним способом на одній стороні аркуша формату А4 з дотриманням встановлених розмірів полів.

Після виконання комп’ютерної реалізації заданої програми, студент доповнює протокол (звіт) результатами її роботи. Звіт оформляється згідно вимог щодо оформлення звітів. Захист комп’ютерного практикуму відбувається на одному із занять. Звіт складається з таких розділів:

1. Назва комп’ютерного практикуму.
2. Мета комп’ютерного практикуму.
3. Теоретичні відомості до виконання комп’ютерного практикуму.
4. Завдання на виконання комп’ютерного практикуму.
5. Блок-схема (для задач комп'ютерних практикумів №4-10).
6. Програма (може бути роздрукованою).
7. Результат роботи програми (у вигляді: вхідні дані-одержані результати).
8. Висновки

Комп’ютерний практикум № 1

СЕРЕДОВИЩЕ РОЗРОБКИ ПРОГРАМ МОВОЮ PYTHON

Мета роботи: Освоєння та практичне оволодіння прийомами роботи в середовищі *Python*, а також набути досвіду підготовки та редагування текстів програм і запуском їх на виконання.

1 Основні теоретичні відомості

Мова програмування *Python* – сучасна високорівнева мова програмування загального призначення, орієнтована на підвищення продуктивності розробки та покращення прозорості програмного коду. Створення мови почалось в 1989 році.

Автор ідеї створення мови *Python* – Ґвідо ван Россум (Guido van Rossum), голландський програміст. Походження назви пов’язують з «Літаючим цирком Монті Пайтона» (Monty *Python's* Flying Circus).

На сьогоднішній день паралельно розвивається 2 гілки (діалекти) версій:

– гілка 2.X останній стабільний реліз 2.7.16 (04 березня 2019 р.);

– гілка 3.X останні стабільні релізи: 3.6.9 (02 липня 2019 р.),
3.7.4 (08 липня 2019 р.).

Діалекти 2.X та 3.X мають обмежену сумісність і використовуються окремо.

Існують різні підходи до створення середовища розробки та застосування програм мовою *Python*:

а) комплектування середовища з окремих модулів «вручну». Такий підхід дозволяє створити мінімально необхідне середовище, що не містить зайвих модулів, не займає зайвого місця і дозволяє працювати на порівняно «слабких» комп’ютерах. Основна частина встановлюється з дистрибутива, який вільно розповсюджується через офіційний сайт https://www.python.org/. Решта необхідних модулів встановлюється інсталятором пакетів *pip* з використанням командного рядка: *pip install modul_name*. Недоліком такого підходу є необхідність самостійно забезпечувати сумісність версій окремих модулів.

б) встановлення одного з інтегрованих дистрибутивів, що поширюються через web-сайти розробників. Як приклад таких дистрибутивів є Anaconda Distribution фірми Anaconda Inc. (https://www.anaconda.com/distribution/) та Canopy фірми Enthought Inc. (https://www.enthought.com/product/canopy/)

Перевагою використання інтегрованих дистрибутивів є включення до їх складу узгоджених версій модулів, що використовуються для наукових розрахунків, а також наявність однієї, або декількох графічних оболонок для створення, тестування та виконання програм. Серед недоліків – досить значні ресурси, що використовуються, а також можливі ускладнення з перенесенням програм на інший комп'ютер, якщо на ньому встановлено інший варіант середовища.

Залежно від способу встановлення та варіанту використання може застосовуватись одне з середовищ розробки/виконання:

– середовище "*Command prompt"* подібне *Python*, (рис. 1.1) придатне для діалогового виконання команд. Характерною особливістю діалогового режиму є знак підказки *">>>"*.

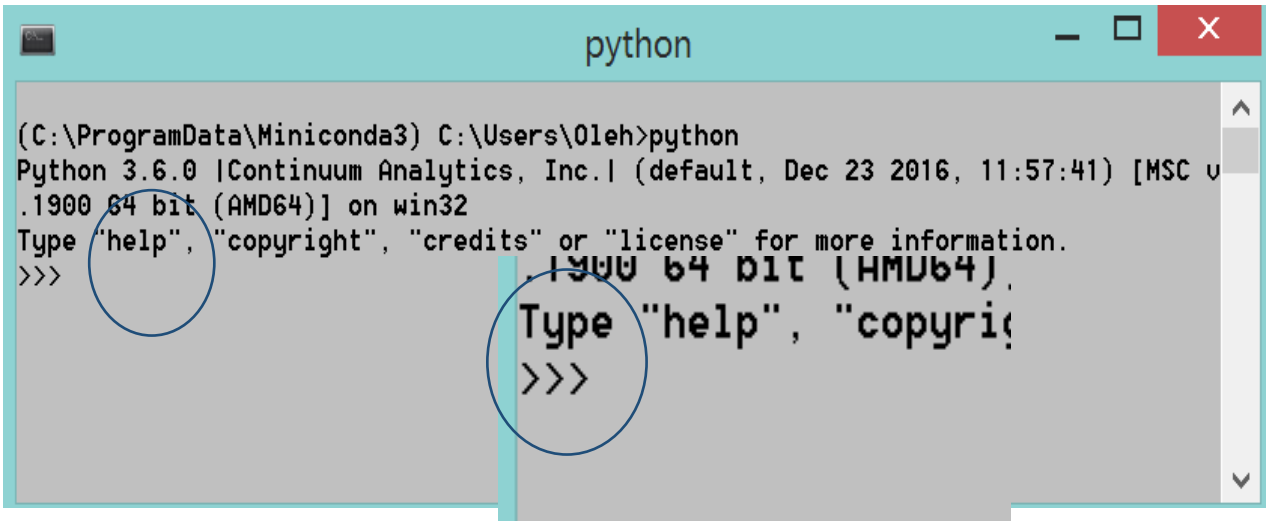


Рисунок 1.1 – "*Command prompt"* подібний режим роботи *Python*

Вказане середовище може бути запущене з командного рядка ("Command prompt") простим вводом команди *Python*. Якщо шлях до директорії,

яка містить файл запуску, не включений до списку стандартних директорій, то попередньо необхідно перейти до неї. Таке середовище дозволяє виконувати окремі команди, створювати та використовувати функції, імпортувати та використовувати стандартні та власні бібліотеки. Виклик бібліотеки:

```
>>>from subprocess import call,
>>>call(["python", "mypy002.py"]).
```

Переваги: висока швидкість запуску та мінімальні вимоги до операційного середовища комп'ютера.

Недоліки: відсутність власного редактора для створення текстів програм.

– середовище *IDLE (Integrated DeveLopment Environment)* – графічна оболонка, яка об'єднує текстові вікна виконання програм (консоль) (рис. 1.2, *а*) та розробки програм (редактор) (рис. 1.2, *б*).

Робота в режимі консолі практично не відрізняється від роботи в середовищі "Command prompt", застосовується той же знак підказки *">>>"*. Очевидною перевагою є те, що режим реалізовано як графічне вікно, з наслідуванням властивостей графічних вікон операційної системи, а значить стає можливим застосування стандартних засобів операційної системи, як наприклад: виділення фрагменту тексту мишею, застосування комбінацій клавіш "Ctrl+C", "Ctrl+V", використання системи меню графічного вікна.

Виконання команди меню вікна консолі "*File -> New file"* (або "*Ctrl+N"*) призводить до створення нового вікна редактора, в якому є можливість створювати, редагувати, зберігати та запускати на виконання тексти програм мовою *Python*. Відкритий таким чином редактор є "інтелектуальним" – розпізнає текст програми під час вводу, забарвлює елементи програми різними кольорами, надає контекстну підказку при роботі з об'єктами.

```
Python 3.6.1 Shell
File  Edit  Shell  Debug  Options  Window  Help
Python 3.6.1 |Anaconda 4.4.0 (64-bit)| (default, May 11 2017, 13:25:24) [MSC v
.1900 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
================================ RESTART: Shell ================================
>>> for i in range(4):
        print(i, ' ',i**2,' ',i**3)


0   0   0
1   1   1
2   4   8
3   9   27
>>> |
```

а

```
*mypy003.py - C:/Users/Oleh/mypy003.py (3.6.1)*
File  Edit  Format  Run  Options  Window  Help
# program for solving of square equation
a=input('Enter A')
b=input('Enter B')|
```

б

а – виконання програм (консоль); б – розробка програм (редактор)

Рисунок 1.2 – Текстові вікна середовища *IDLE* (*Python 3.6.1*, у складі пакету *Anaconda 4.4.0*)

Програму, текст якої створено в вікні редактора або завантажено в нього через меню, можна запустити на виконання, і тоді результати її роботи будуть відображатися у вікні консолі.

Переваги: порівняно висока швидкість запуску та мінімальні вимоги до операційного середовища комп'ютера, доступність значної кількості команд віконного режиму, наявність інтелектуального редактора для створення програм, застосування єдиного інтерфейсу для вікон консолі та редактора.

– середовище *IPython* – інтерактивний *Python* – розвиток та вдосконалення "Command prompt" середовища, який поєднує його можливості та окремі можливості командної оболонки *Unix*, надає засоби редагування коду і візуалізації даних та іншу функціональність. *IPython* взаємодіє з великою кількістю графічних бібліотек, широко застосовується в бібліотеках наукового

призначення. Характерним для *IPython* є застосування в якості підказки фрази, шо складається з слова *In* – введення або *Out* – виведення та цілого числа, взятого в квадратні дужки (*In* [2]).

В чистому вигляді середовище *IPython* на сучасних комп'ютерах застосовують рідко, але на його основі чи з його застосуваннями розроблено більш сучасні інтегровані середовища – *Spyder* та *Jupyter*.

– середовище *Spyder* – повнофункціональне кросплатформене інтерактивне середовище, оптимізоване для проведення наукових розрахунків. Вікно оболонки *Spyder* (рис. 1.3) включає поле програми (1), поле перегляду (2) з закладками *Variable explorer, File explorer і Help*, а також поле консолі (3) з закладками *Python console*, *History log* і *IPython console*. Фактично засобами робочого вікна реалізується практично весь функціонал застосування *Python,* як для програмування, так і для інтерактивних розрахунків. Крім того, текстовий редактор програм, вбудований у *Spyder* має можливість підсвічувати синтаксис не тільки *Python* але й *C/C++* та *Fortran*.

Недоліки середовища *Spyder* викликані його універсальністю і полягають у тому, що його робота вимагає досить значних ресурсів комп'ютера, а час старту програми, навіть на сучасних системах може складати від 1 до 2 хв.

– середовище *Jupyter* – сучасний розвиток *IPython*, що реалізує кросплатформенну роботу з відображенням інформації через стандартний інтернет-браузер (рис. 1.4). *Jupyter* використовується як одне з основних середовищ інтерактивної роботи в комплектаціях пакетів *Python* спеціалізованих для наукових розрахунків.

Jupyter об'єднує можливості створення програм, функцій, бібліотек; їх запуск і перезапуск, повторне виконання коду, створення коментарів, у тому числі ілюстрованих, використання ілюстрацій, створених програмою, виконання команд оболонки операційної системи та багато іншого. Фактично: результат програмування в середовищі *Jupyter* – блокнот, покрокове виконання якого з підстановкою даних забезпечує одержання результату. Досить часто роботу в

Jupyter порівнюють з використанням потужної пропрієтарної системи діалогової математики *MatLab*.

Недоліки середовища *Jupyter* полягають у необхідності значних ресурсів комп'ютера.

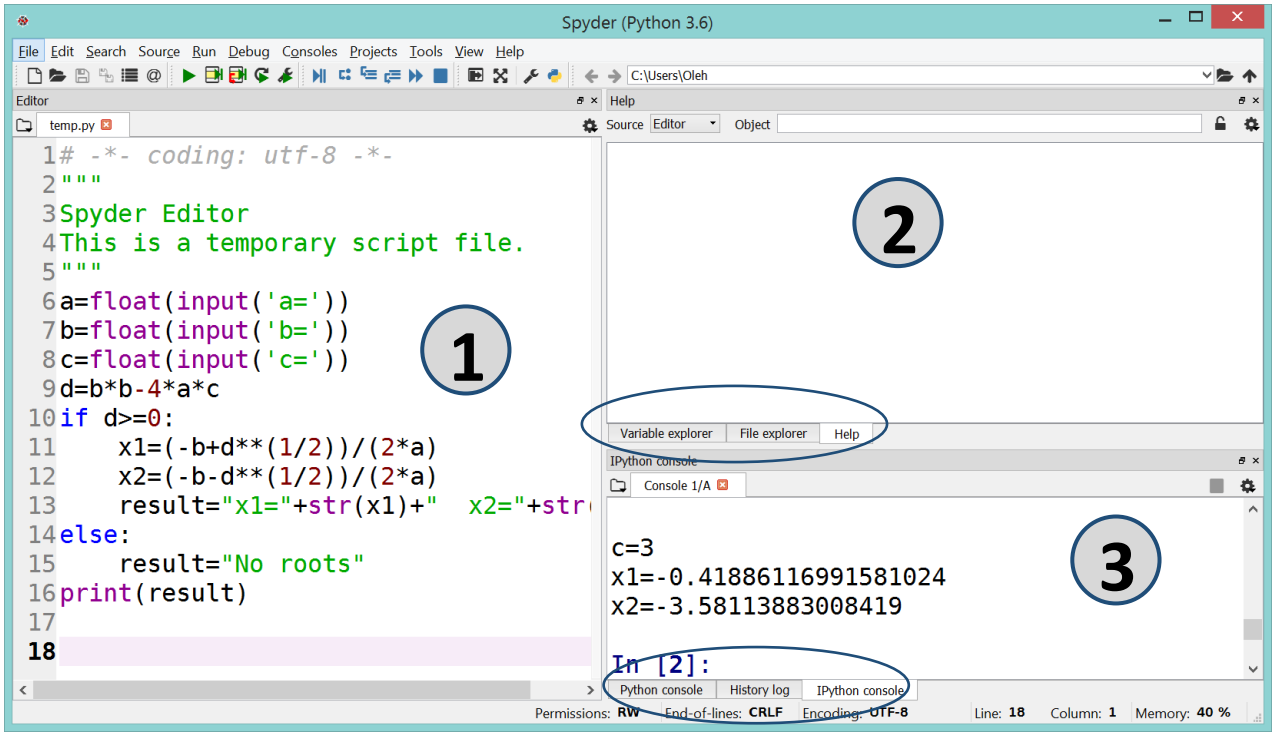


Рисунок 1.3 – Робоче вікно середовища *Spyder*

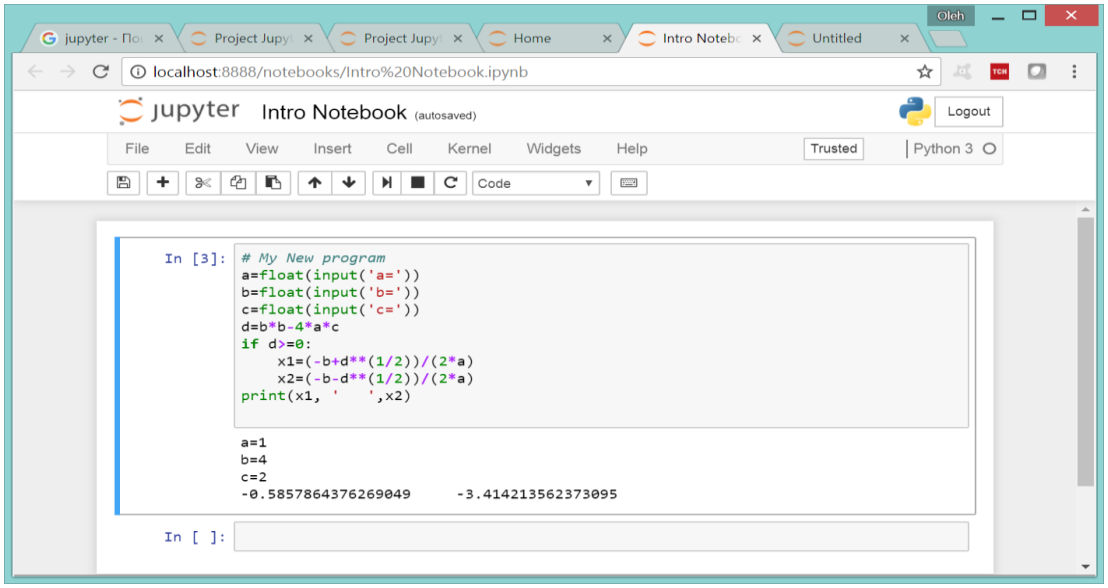


Рисунок 1.4 – Середовище *Jupyter* в браузері *Google Chrome*

2 Проведення розрахунків

Будь-яке з інтерактивних середовищ *Python* – консольне з підказкою ">>>" чи типу *IPython* з підказкою "[1]" передбачає можливість вільного проведення математичних розрахунків. Обчислення можуть проводитись як із застосуванням змінних, так і безпосередньо з числами. Змінні в мові *Python* не декларуються, але обов'язково ініціалізуються перед використанням:

```
>>> a=2
>>> b=7
>>> b/a+c
Traceback (most recent call last):
  File "<pyshell#27>", line 1, in <module>
    b/a+c
NameError: name 'c' is not defined
>>>
>>> c=3
>>> b/a+c
6.5
>>>.
```

Оскільки до моменту використання в розрахунках змінну **c** не було ініційовано (їй не присвоєно значення), то система згенерувала помилку : *name 'c' is not defined.* Після ініціалізації змінної обчислення проведено успішно з виведенням результату: 6.5. Навіть без використання змінних, одержаний результат можна використовувати в подальших розрахунках:

```
6.5
>>>_*3
19.5.
```

Символ підкреслювання викликає останнє одержане значення.

Під час проведення розрахунків застосовуються наступні знаки математичних операцій:

" + " додавання

" - " віднімання

" * " множення

“ / “ ділення*

“ // “ цілочисельне ділення*

“ % “ залишок від цілочисельного ділення

“ ** “ степінь

* запис операцій ділення відрізняється в реалізаціях python 2.X та 3.X. Так, у версії 2.X знак “ / “ реалізує контекстно-орієнтоване ділення (залежне від операндів). Якщо і ділене і дільник представлено цілими числами, то реалізується операція цілочисельного ділення. Якщо хоча б один операнд записано як дійсне число (в числове значення включено символ ” . ”, або застосовано функцію перетворення типів), то реалізується звичайне ділення.

Крім операндів (числових значень чи ініційованих змінних) та знаків математичних операцій в розрахунках можуть використовуватись вбудовані функції, функції зовнішніх бібліотек та функції користувача. Створення та застосування функцій користувача а також опис зовнішніх бібліотек буде розглянуто пізніше.

Оскільки базова версія *Python* не є орієнтованою на розрахунки, то перелік вбудованих функцій математичного призначення обмежується функціями:

– *abs (x)* – модуль числа x

– *divmod(x, y)* – частка та залишок цілочисельного ділення x на y.

Корисною може бути функція "*type( )"*, що повертає тип аргументу. Докладніше про типи даних в *Python* розглянуто в комп'ютерному практикумі № 2.

3 Розроблення елементарної програми

Характерною особливістю програми мовою *Python* є відсутність знаків чи ключових слів, що обмежують програмний блок – таких як фігурні дужки в мові *C*, або логічні дужки *Begin – End* в мові *Pascal*. Програмний блок розміщується з відступом з лівого боку, для чого може використовуватись клавіша *Tab*. Більшість інтегрованих середовищ розробки автоматично створюють такий відступ, якщо попередній рядок програми закінчується двокрапкою « : ». Після закінчення

програмного блоку, точку введення (курсор) необхідно повернути в правильну позицію.

Для практичного оволодіння процесом створення тесту програми необхідно ввести в середовищі IDLE наступні варіанти програми для розв'язання квадратного рівняння.

Варіант 1. Звичайна програма для пошуку та виводу дійсних коренів квадратного рівняння:

```
# program for solving of square equation
a=float(input("a="))
b=float(input("b="))
c=float(input("c="))
d=b**2-4*a*c
if d<0:
    ss='No real roots'
else:
    x1=(-b+d**(1./2))/(2*a)
    x2=(-b-d**(1./2))/(2*a)
    ss='x1=' + str(x1)+',   x2='+str(x2)
print(ss)
```

Запуск програми здійснюється безпосередньо з інтегрованого середовища розробки.

Варіант 2. Функція пошуку дійсних коренів квадратного рівняння з поверненням комплекту коренів у вигляді кортежу – tuple (специфічний тип даних):

```
# function for solving of square equation
def func_sqw(a,b,c):
    d=b**2-4*a*c
    if d<0:
        answ=False
    else:
        x1=(-b+d**(1./2))/(2*a)
        x2=(-b-d**(1./2))/(2*a)
        answ=(x1,x2)
    return(answ)
```

Запуск програми здійснюється з інтегрованого середовища розробки, після чого в *Python Shell* необхідно викликати функцію, підставивши коефіцієнти квадратного рівняння, наприклад:

```
>>> func_sqw(1, 5, 3)
або
>>>roots=func_sqw(2, 8, 1)
>>>print(roots[0], roots[1]).
```

Варіант 3. Програма для пошуку та виводу коренів квадратного рівняння, включаючи комплексні.

```
# program for solving of square equation
a=float(input("a="))
b=float(input("b="))
c=float(input("c="))
d=b**2-4*a*c
x1=(-b+d**(1./2))/(2*a)
x2=(-b-d**(1./2))/(2*a)
if d<0:
    root_type='complex'
else:
    root_type='real'

print('The equation has ', root_type, ' roots')
ss='x1=' + str(x1)+',   x2='+str(x2)
print(ss)
print('Control')
print(str(x1), 'a*x**2 + b*x + c =', str(a*x1**2+b*x1+c))
print(str(x2), 'a*x**2 + b*x + c =', str(a*x2**2+b*x2+c))
```

Запуск програми здійснюється як у варіанті 1.

4 Завдання на комп'ютерний практикум

Реалізувати в середовищі *IDLE* задані викладачем варіанти програми. Для кожного варіанту випробувати роботу програми не менше ніж на 5 комплектах коефіцієнтів квадратного рівняння, підібраних таким чином, щоб дискримінант рівняння мав позитивне значення, значення рівне нулю та негативне значення. Занотувати одержані результати. Зробити висновки та записати їх у протокол роботи.

5 Запитання для самоконтролю

1. Чи є *Python* спеціалізованою мовою програмування?
2. Яким чином в програмі мовою *Python* виділяються програмні блоки?
3. Чи існують інтегровані середовища розробки програм мовою *Python*?

Література: [1] С. 63 – 110. [2] С. 6 – 35. [3] С. 7 – 20.

Комп’ютерний практикум № 2

СТРУКТУРА ПРОГРАМИ МОВОЮ PYTHON. ВВІД - ВИВІД ДАНИХ. ЛІНІЙНІ ОБЧИСЛЕННЯ. БІБЛІОТЕКА MATH

Мета роботи: здобуття практичного досвіду підготовки та редагування текстів в мові програмування *Python*; використання команд вводу-виводу та математичних функцій бібліотеки *math*.

1 Основні теоретичні відомості

1.1 Структура програми мовою *Python*

На відміну від значної кількості мов програмування, *Python* має порівняно просту структуру програми, місце окремих елементів чітко не визначено.

Для невеликих програм необхідно дотримуватись простого правила: ***Перед використанням об'єкт повинен бути ініційований***. Ця вимога має різне значення для різних об'єктів.

Змінні мають бути ініційовані шляхом присвоєння їм початкового значення. Поява змінної, яка не має значення, в лівій частині виразу присвоювання або в ролі аргументу функції викликає зупинку програми і повідомлення типу:

```
Traceback (most recent call last):
    File "C:/Users/.../p2-1.py", line 2, in <module>
    c = d*a
NameError: name 'a' is not defined.
```

Останній рядок повідомляє, що змінна з іменем 'a' не визначена.

Функції мають бути описані перед своїм першим використанням. Якщо функцію не описано, або описано в тексті програми нижче ніж міститься її виклик, програма зупиняється з повідомленням аналогічним вище наведеному.

Бібліотеки – попередньо підготовлені набори текстів програм, описів даних, значень, функцій, тощо – мають бути завантажені командою *import* до першого використання їх елементів.

Для більш зручного читання тексту програми, може бути рекомендована наступна структура програми:

– **коментар** – текстовий блок, який описує роботу програми, порядок і тип введення даних, тощо. Однорядковий коментар позначається знаком # у лівій позиції рядка. Багаторядковий коментар позначається потрійними лапками або апострофами перед і після текстового блоку, до прикладу:

```
'''
This is a comment, which
use more then one string.
It must be separated from program text
'''
```

– **команди завантаження бібліотек** – за потреби використання зовнішніх бібліотек та їх елементів (докладніше з прикладами це описано в п. 1.4)

– **описи функцій користувача** – за потреби. У функції користувача переносять частини програми, які виконуються декілька разів та / або мають ізольоване призначення.

– **текст програми** – основний текст програми, який виконує відповідну роботу, і де можуть використовуватись описані вище функції користувача та елементи завантажених бібліотек.

У кожному з структурних елементів, крім коментарів, діє правило: один оператор займає один рядок. Якщо оператору необхідно продовжити запис на декілька рядків, для цього застосовують спеціальний синтаксис:

– використання символу оберненої косої риски "\" у кінці рядка який потребує перенесення:

```
volume = lenght * width\
    *height.
```

– використання дужок (круглих, квадратних або фігурних) залежно від ситуації:

```
sdf=[1,2,3,4,5,6,7,8
    ,9,10,11,12]
```
перенесення всередині квадратних дужок,

```
print(sdf[1], sdf[2], sdf[3],
    sdf[4], sdf[5]
```
перенесення всередині круглих дужок.

Необхідно пам'ятати, що програмні блоки, які є тілом циклів (розглянуто в комп'ютерному практикумі № 4), умовних операторів (розглянуто в комп'ютерному практикумі № 5), функцій (розглянуто в комп'ютерному практикумі № 6) описів об'єктів вводяться зі стандартним відступом від попереднього тексту. Відступ формується автоматично більшістю інтегрованих середовищ для створення програм.

ВАЖЛИВО: У мові програмування *Python* розрізняються великі та малі літери. Змінна **ABc** і змінна **aBc** є різними змінними з різними значеннями.

Рекомендації щодо оформлення коду програм вказані в офіційному Style Guide: https://www.python.org/dev/peps/pep-0008/.

1.2 Ввід-вивід даних

Інструкцією вводу даних в *Pyton* є *input*, однак повний набір засобів вводу залежить від версії *Python.* У версії 2.7.X існують інструкції *input( )* та *raw_input()*. У версії 3.X інструкція *raw_input( )* замінена інструкцією *input( )*, а інструкція *input( )* може бути відтворена через *eval(input( )).*

Інструкція *raw_input( )* (*Python 2.7*) призначена для введення рядка символів, тому при виконанні коду програми:

```
a=raw_input(" <= a ")
b=raw_input(" <= b ")
print(a+b)
```

буде одержано результат

```
>>>
<= a 723
<= b 754
723754.
```

Схожий результат одержується при використанні інструкції *input( )* у версії *Python* 3.X.

Якщо програмою передбачено введення числових даних, то у версії 3.X варто використовувати один з наведених варіантів:

1. *a = eval(input(" a = "))* - функція *eval( )* автоматично визначає тип числового значення;

2. *b = int(input(" b = "))* - функція *int( )* сприймає введене як ціле число. Але якщо буде введене число з плаваючою десятковою крапкою, то система згенерує помилку.

3. *d = float(input(" d = "))*- функція *float( )*сприймає введене як дійсне число (число з плаваючою десятковою крапкою).

Операція виводу здійснюється за допомогою інструкції *print*, яка в різних версіях *Python* також має свої особливості. Універсальний варіант інструкції (працює в обох версіях) має вигляд:

```
print('String for output'),
```

де замість `'String for output'` знаходиться простий чи складений рядковий об'єкт:

```
print('A= ' + str(a)+' B= '+str(b))
print("No real roots")
print("a="+str(a)+"\n"+"b="+str(b)+"\n"+"a+b="+str(a+b))".
```

Під час виведення окремих значень функцію перетворення до рядкового типу *str()* можна опустити.

1.3 Лінійні математичні обчислення

Використання в програмі лінійних математичних обчислень є одним з найпростіших способів програмування. Обчислення в програмі виконуються застосуванням оператора присвоювання, який являє собою комбінацію імені змінної, знаку дорівнює **' = '**, та виразу, чиє значення має бути присвоєне змінній:

$$a = 4*c + 2*d**2 - 2/v*f.$$

Математичні операції у виразі виконуються згідно правил виконання математичних операцій та їх пріоритетів. Тому наведений рядок символів відповідає наступній математичній формулі:

$$a = 4c + 2d^2 - \frac{2}{v}f. \tag{1.1}$$

Усі знаки математичних операцій повинні бути вказані явно. Якщо значення змінних, що використані у виразі буде змінено після даного оператора присвоювання, то це не буде мати впливу на одержаний у ньому результат – обчислення в програмі здійснюються згори-донизу, в рядку зліва-направо з врахуванням пріоритетності операцій.

Якщо змінній присвоюється нове значення – своє попереднє значення вона "забуває" назавжди.

З метою спрощення роботи з текстом програми рекомендується, щоб вираз в операторі присвоювання мав "розумну" довжину, не перевищував видиму частину вікна редактора. Для цього зручно розділяти вираз на складові і записувати в декількох операторах присвоювання, в окремому рядку кожен.

В мові *Python* оператор присвоювання може бути багатомісним:

```
>>> a, b= 5, 7
>>> a
5
>>> b
7
>>> c, d = 2*a+3*b, 3*a+2*b
>>> c
31
>>> d
29.
```

Використовувати такі багатомісні оператори присвоювання необхідно обґрунтовано, оскільки вони погіршують можливість читання та розуміння тексту програми.

Окремим видом оператора присвоювання є скорочений оператор:

qw1+=2 збільшує значення *qw1* на 2

qw3-=7 зменшує значення *qw3* на 7

qw5=2* збільшує значення *qw5* у 2 рази

qw6/=3 зменшує значення *qw6* в 3 рази

*qx3**=4* піднімає значення *qx3* до 4-ї степені

1.4 Бібліотека *math*

Як зазначалось в практикумі 1, мова *Python* без підключення зовнішніх бібліотек практично не містить математичних функцій. У випадку використання програми для математичних розрахунків застосовують бібліотеку *math*. Формальний опис бібліотеки *math* міститься за адресою: *https://docs.python.org/3/library/math.html.*

Бібліотека *math*, входить до стандартного комплекту *Python*, встановлюється і доступна на усіх системах. Ця бібліотека за набором математичних функцій та синтаксисом їх застосування збігається з відповідними стандартними бібліотеками для мов *C* та ін.

Бібліотека *math* містить набір математичних функцій віднесених до 5 різних типів і призначених для роботи з дійсними числами (числами з плаваючою десятковою крапкою). Для відповідної обробки комплексних чисел існує окрема бібліотека *cmath*. Крім опису функцій, бібліотека *math* містить набір найважливіших математичних констант.

Для використання в програмі елементів бібліотеки *math*, бібліотека, або її окремі елементи мають бути завантажені командою *import* одним з перерахованих способів:

а) import math – завантажує для використання в програмі весь вміст бібліотеки *math*, після чого елемент бібліотеки (функції чи константи) може бути викликаний в програмі з синтаксисом:

math.sin(a) – вказується ім'я бібліотеки і, через крапку, ім'я функції;

б) import math as mt – завантажує для використання в програмі весь вміст бібліотеки *math* і надає їй коротке ім'я (спосіб рекомендований для бібліотек з

складними іменами, що містять багато символів). У цьому випадку елемент бібліотеки може бути викликаний в програмі з синтаксисом:

mt.cos(a) – вказується скорочене ім'я бібліотеки і, через крапку, ім'я функції;

в) from math import pi, sin, cos – завантажує для використання в програмі тільки перераховані елементи, які викликаються безпосередньо, без вказування імені бібліотеки:

*x=sin(pi/4)**2+cos(pi/4)**2.*

г) from math import * – завантажує усі елементи бібліотеки, якщо пряме завантаження окремих з них не заперечується внутрішньою структурою бібліотеки. Елементи, як і в попередньому випадку стають доступними напряму, без вказування імені бібліотеки. Такий спосіб рекомендують тільки для досить обмеженої кількості бібліотек, які мають не дуже великий розмір і прозору структуру, в іншому випадку може виникнути ситуація конфлікту імен зі змінними, які використовуються в програмі. Список функцій і констант модуля *math* наведено в таблиці 2.1.

Таблиця 2.1 – Список функцій і констант модуля *math*

Функція	**Тип аргументу (ів)**	**Тип результату**	**Призначення**
1	2	3	4
Функції теорії чисел та представлення чисел			
***math*.ceil(x)**	float	integer	Найменше ціле, що більше або дорівнює **x**
***math*.copysign(x, y)**			Надає **x** знак **y**
***math*.fabs(x)**	numerical	numerical	Абсолютне значення (модуль) числа
***math*.factorial(x)**	Позитивне ціле	ціле	Факторіал числа
***math*.floor(x)**	float	integer	Найбільше ціле, що менше або рівне **x**
***math*.fmod(x, y)**	Два числа	дійсне	Залишок від ділення **x** на **y**. Має підвищену точність при роботі з дійсними числами

Продовження таблиці 2.1

1	2	3	4
***math*.frexp(x)**	numerical	Дійсне і ціле	Мантиса і степінь для представлення числа у вигляді **x=m*2**e**
***math*.fsum(*iterlist*)**			Аналог функції **sum**, але з вищою точністю
***math*.factorial(x)**	Позитивне ціле	ціле	Факторіал числа
***math*.floor(x)**	float	integer	Найбільше ціле, що менше або рівне **x**
***math*.fmod(x, y)**	Два числа	дійсне	Залишок від ділення **x** на **y**. Має підвищену точність при роботі з дійсними числами
***math*.frexp(x)**	numerical	Дійсне і ціле	Мантиса і степінь для представлення числа у вигляді **x=m*2**e**
***math*.fsum(*iterlist*)**			Аналог функції **sum**, але з вищою точністю
***math*.gcd(a, b)**	Два цілих	Ціле	Повертає найбільший спільний дільник
***math*.isclose(a,b)**	Два дійсних	boolean	Перевіряє чи достатньо близькі значення двох змінних. (має додаткові аргументи)
***math*.isfinite(x)**		boolean	Перевіряє чи є аргумент скінченим числом
***math*.isinf(x)**	numerical	boolean	Перевіряє, чи є число +/- нескінченністю
***math*.isnan(x)**		boolean	Перевіряє, чи аргумент NaN (not a number)
***math*.ldexp(x, i)**	Дійсне і ціле	Дійсне	Повертає: **z=x*(2**i).** Обернено до ***math*.frexp**
***math*.modf(x)**	Дійсне	Два дійсних	Повертає кортеж з двох дійсних чисел, що є дробовою та цілою частиною **x**
***math*.trunc(x)**	Дійсне	Дійсне	Повертає число, обрізане до цілого
***math*.exp(x)**	Дійсне	Дійсне	Експонента числа **x, e^x**
***math*.expm1(x)**	Дійсне	Дійсне	**exp(x)-1** для збереження точності при малих **x**

Продовження таблиці 2.1

1	2	3	4
Степеневі та логарифмічні функції			
math.**log(x [,base])**	Дійсне, base - позитивне	Дійсне	Натуральний логарифм або логарифм за основою **base**, якщо вказана
math.**log1p(x)**	Дійсне	Дійсне	Натуральний логарифм від **(1+x)** для збереження точності при малих **x**
math.**log2(x)**	Дійсне	Дійсне	Логарифм за основою 2, з підвищеною точністю
math.**log10(x)**	Дійсне	Дійсне	Логарифм за основою 10, з підвищеною точністю
math.**pow(x, y)**	Два дійсних	Дійсне	Степінь числа **x**y** з підвищеною точністю
math.**sqrt(x)**	Дійсне, позитивне	Дійсне	Квадратний корінь
Тригонометричні функції			
math.**asin(x)**	Дійсне	Дійсне	Арксинус аргумента
math.**acos(x)**	Дійсне	Дійсне	Арккосинус аргумента
math.**atan(x)**	Дійсне	Дійсне	Арктангенс аргумента
math.**atan2(x, y)**	Два дійсних	Дійсне	Арктангенс кута, визначеного двома катетами прямокутного трикутника
math.**cos(x)**	Дійсне	Дійсне	Косинус кута в радіанах
math.**hypot(x, y)**	Два дійсних	Дійсне	Евклідова норма. **sqrt(x*x+y*y)**
math.**sin(x)**	Дійсне	Дійсне	Синус кута в радіанах
math.**tan(x)**	Дійсне	Дійсне	Тангенс кута в радіанах
Функції перетворення кутів			
math.**degrees(x)**	Дійсне	Дійсне	Перетворення радіанів в градуси
math.**radians(x)**	Дійсне	Дійсне	Перетворення градусів в радіани
Гіперболічні функції			
math.**acosh(x)**	Дійсне	Дійсне	Обернений гіперболічний косинус
math.**asinh(x)**	Дійсне	Дійсне	Обернений гіперболічний синус
math.**cosh(x**	Дійсне	Дійсне	Гіперболічний косинус
math.**atanh(x)**	Дійсне	Дійсне	Обернений гіперболічний тангенс
math.**sinh(x)**	Дійсне	Дійсне	Гіперболічний синус
math.**sinh(x)**	Дійсне	Дійсне	Гіперболічний тангенс

Продовження таблиці 2.1

1	2	3	4
***math.*acosh(x)**	Дійсне	Дійсне	Обернений гіперболічний косинус
***math.*asinh(x)**	Дійсне	Дійсне	Обернений гіперболічний синус
***math.*atanh(x)**	Дійсне	Дійсне	Обернений гіперболічний тангенс
***math.*cosh(x**	Дійсне	Дійсне	Гіперболічний косинус
***math.*sinh(x)**	Дійсне	Дійсне	Гіперболічний синус
***math.*sinh(x)**	Дійсне	Дійсне	Гіперболічний тангенс
Спеціальні функції			
***math.*gamma(x)**	Дійсне	Дійсне	Гамма-функція від x
***math.*lgamma(x)**	Дійсне	Дійсне	Натуральний логарифм гамма-функції від x
Константи			
math.pi	**3.141592653589793**	Число **π**	
math.e	**2.718281828459045**	Основа натурального логарифму	
math.tau	**6.283185307179586**	Число **2π**	
math.inf		Нескінченність у форматі дійсного числа	
math.nan		"Не число" у форматі дійсного числа	

2 Завдання на комп'ютерний практикум

1. У діалоговому режимі роботи з оболонкою *Python Shell*:

– провести прості обчислення без присвоювання результату змінній;

– використовувати символ виклику попереднього результату;

– провести прості обчислення з застосуванням оператора присвоювання;

– опрацювати застосування багатомісного оператора присвоювання;

– опрацювати застосування скороченої форми оператора присвоювання;

– опрацювати різні способи виклику та використання бібліотеки *math*.

Для зміни режиму імпорту бібліотеки необхідно проводити перезапуск діалогової оболонки (пункт меню *"Shell ->Restart Shell"*, або комбінація клавіш *"Ctrl+F6"*).

2. У режимі програмування створити, відлагодити та виконати програму для проведення розрахунків за формулами, наведеними в Додатку 2.

3 Запитання для самоконтролю

1. В якому форматі передає у програму дані оператор вводу *input* у версії *Python 3.X*?

2. Які функції округлення до цілого числа існують в бібліотеці *math*?

3. Який знак застосовують для продовження запису оператора програми на декілька рядків?

4. Які значення послідовно виведе наступна програма?

```
a, b, e = 7, 8, 3
print(e)
from math import *
print(e)
e+=a
print(e).
```

5. Як засобами *Python* (за допомогою бібліотеки *math* або без неї) записати наступні вирази ?

$t = e^p + \sqrt[3]{g}$

$k = ln5 + log_x y$

$m = 3cos^2 y - 2sin^2 x$

$z = tgx + ctgy$

$$a = ln\left|y - \sqrt{|x|} \cdot \left(x - \frac{y}{z + \frac{x^2}{4}}\right)\right|$$

6. Як одержати значення виразу в градусах?

$\beta = \mathrm{arctg}(1)$

Література: [1] С. 323 – 374. [2] С. 36 –81. [3] С. 21 –33.

Комп’ютерний практикум № 3

ТИПИ ДАНИХ В PYTHON

Мета роботи: освоєння типів даних, правил їх використання, оволодіння засобами перетворення типів даних та приведення до заданого типу.

1 Основні теоретичні відомості

1.1 Скалярні типи даних в *Python*

В процесі розміщення в пам'яті комп'ютера даних, програмі чи програмній оболонці необхідна інформація про їх тип для коректної обробки, обчислення кількості місця та організації доступу. Переважна більшість мов програмування, особливо компілюючих, вимагає обов'язкове декларування змінних до їх використання з вказуванням імені змінної та імені типу.

За роботою зі змінними, мова програмування *Python* значно відрізняється від описаного, – у ній не передбачене декларування змінних. Тип призначається змінній автоматично, коли вона одержує значення і може змінюватись протягом виконання програми. Наприклад, виконання перерахованих команд в *Python Shell* дозволяє змінній *a* приймати послідовно значення типів: рядкового – *'str'*, цілого – *'int'*, дійсного (з плаваючою десятковою крапкою - *'float'*)та булевого – *'bool'*.

```
>>> a=input('A=')
A=17
>>> a, type(a)
('17', <class 'str'>)
>>> a=int(a)
>>> a, type(a)
(17, <class 'int'>)
>>> a=float(a)
>>> a, type(a)
(17.0, <class 'float'>)
>>> a=bool(a)
>>> a, type(a)
(True, <class 'bool'>).
```

Фактично, перераховані типи являють собою не традиційні для програмування типи даних, а об'єкти відповідних класів. Об'єкт визначеного класу створюється в момент виконання операції присвоювання, при цьому відповідному ідентифікатору (наприклад *a* в попередньому прикладі) приписується не саме значення, а адреса відповідного об'єкта в пам'яті комп'ютера. Створене значення залишається на тому ж місці (за тією ж адресою) до тих пір, поки на нього вказує хоча б одна змінна. Процес переприсвоювання відбувається згідно схеми, показаної на рисунку 3.1.

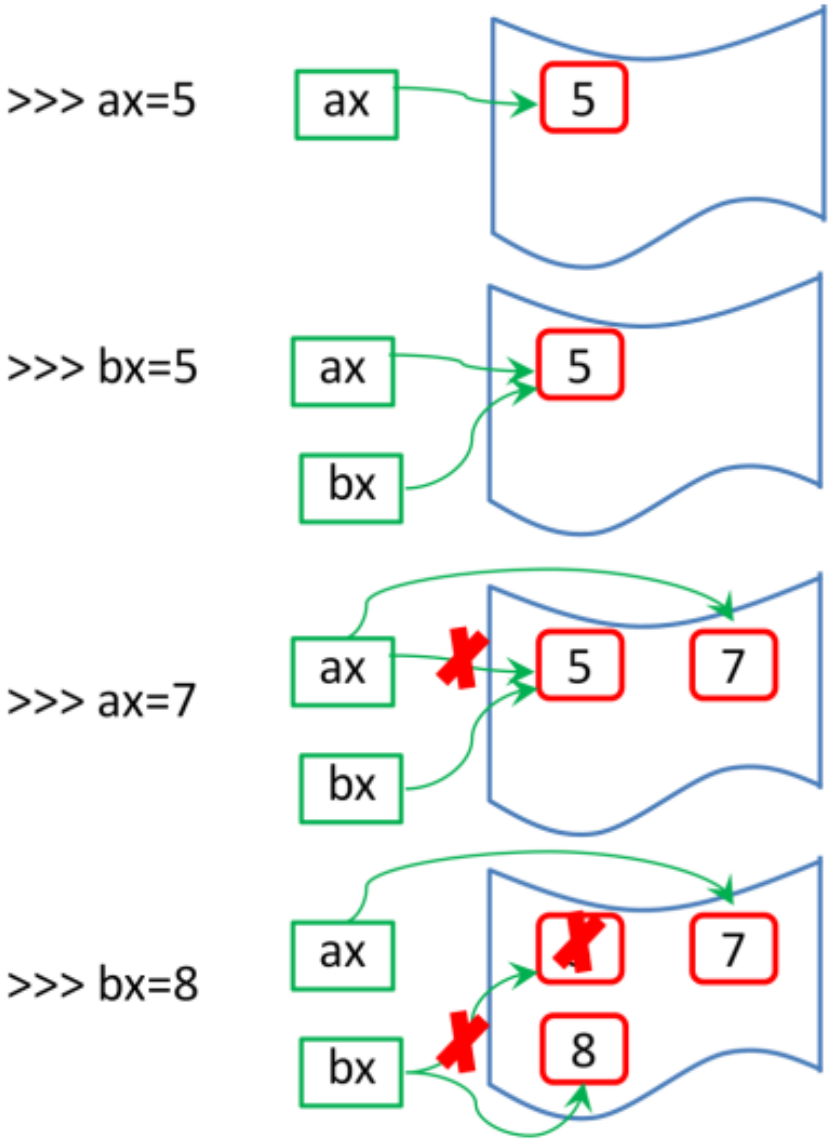


Рисунок 3.1 – Схема присвоєння вказівників на значення в мові *Python*

Основними скалярними класами (типами даних) в *Python* є:

– цілочисельні, до яких відноситься клас *int* та клас *long*. Клас *int* використовується, якщо змінна одержує цілочисельне значення в межах від –2147483647 до +2147483647 (що відповідає значенню $\pm 2^{31}$-1. Клас *long* приписується, якщо цілочисельне значення виходить за вказані межі. У версії *Python 3.X* клас *long* відсутній, а межу значень цілого типу не вказано. Цілочисельні значення зберігаються у пам'яті комп'ютера у вигляді чисел, записаних в двійковій системі числення – у вигляді послідовності нулів та одиниць.

– дійсні (з плаваючою десятковою крапкою) – клас *float*. Цей клас є аналогом типів *double* у більшості інших мов програмування. Для зберігання значень цього типу використовується 8 байтів = 64 біти. Тип *float* приписується при присвоєнні числового значення, яке містить десяткову крапку (*3.14; 2.3; .15; 31.*)

– комплексні – клас *complex*. Клас служить для збереження та застосування комплексних чисел, що є сумою дійсної частини (*real part*) та уявної частини (*imaginary part*). Кожна з частин записується окремим числом цілим або дійсним, уявна частина закінчується знаком *j* або *J*, що символізує множник $\sqrt{-1}$. Комплексні числа обробляються за правилами алгебри комплексних чисел.

– булеві – клас *bool***.** Клас передбачає наявність двох значень: *True* – істина та *False* – неправда. Значення змінній можуть бути як прямим присвоюванням (обов'язково з дотриманням написання через велику літеру), так і як результат булевої операції.

Числові значення кожного з класів записуються за допомогою відповідних літералів.

Так цілочисельні значення записуються в одному з варіантів: як десяткові – звичні для читання числа, як шістнадцятирічні – починаються з знаків 0x або 0X (нуль та літера X) і містять знаки від 0 до 9 та від A до F (літери можуть застосовуватись як великі так і малі), восьмеричні – починаються зі знаків 0o або 0O (нуль та літера O) і містять знаки від 0 до 7, двійкові – починаються зі знаків

0b або 0B (нуль і літера b) і містять знаки 0 або 1. Присвоювати значення можна у будь-якому форматі, зберігаються вони завжди в двійковому вигляді, а відображаються – в десятковому, як показано в прикладі:

```
>>> a=5
>>> b=0b111011
>>> c=0o7237
>>> d=0xFA12
>>> a,b,c,d
(5, 59, 3743, 64018)
>>> a+b+c+d
67825.
```

Для перегляду числа у форматі відмінному від десяткового застосовують функції – модифікатори вигляду: *hex(x)* – представлення в шістнадцятирічному форматі, *oct(x)* – представлення у восьмирічному форматі, *bin(x)* – представлення в двійковому форматі.

```
>>>ast=bin(a)  #a=5
>>>bst=oct(b)  #b=0b111011  (59)
>>>cst=hex(c)  #c=0o7237    (3743)
>>> ast, bst, cst
('0b101', '0o73', '0xe9f').
```

Однак потрібно розуміти, що результатом роботи цих функцій є рядок символів – як помітно з наведеного результату виводу. Для повернення до числового формату необхідно застосовувати функцію *int* у форматі *int(str, base):*

```
>>> int(ast, 2)
5
>>> int(bst, 8)
59
>>> int(cst, 16)
3743
>>>.
```

Цілі числа є об’єктами – екземплярами класу *int*. Як об’єкти вони мають ряд властивостей (аналог функцій) та методів (аналог значень змінних):

bit_length – метод, що повертає кількість бітів, необхідні для запису числа;

conjugate – метод, повертає число, спряжене з даним комплексним;

numerator – дескриптор, чисельник при представленні числа натуральним дробом;

denominator – дескриптор, знаменник при представленні числа натуральним дробом;

imag – дескриптор, уявна частина комплексного числа;

real – дескриптор, дійсна частина комплексного числа;

from_bytes – метод, перетворює послідовність байтів у число;

to_bytes – метод, перетворює число в послідовність байтів.

Значення з плаваючою десятковою крапкою – об’єкти класу *float* записуються в одному з форматів:

a=3.6754 звичний вигляд числа з явним записом цілої та дробової частин, розділених десятковою крапкою;

b=.72 вигляд числа з опущеним лідируючим нулем (b=0.72);

c=1.32e+5 експоненційний вигляд числа – записано декілька значущих цифр та степінь десяти, на яку необхідно домножити вираз (c=132000);

d=17. варіант запису цілого числа у вигляді числа з плаваючою крапкою.

В пам’яті комп’ютера числа *float* зберігаються в апроксимованому вигляді – як набір чисельника та знаменника, що при діленні дають число, достатньо близьке до заданого.

Властивості та методи класу *float:*

conjugate – теж, що для чисел *int;*

as_integer_ratio – метод, виводить кортеж з чисельника та знаменника;

fromhex – запис числа *float* в шістнадцятирічному форматі;

hex – перетворення шістнадцятирічного запису в число *float;*

imag – дескриптор, аналог для *int;*

real – дескриптор, аналог для *int;*

is_integer – метод, повертає булеве значення *True,* якщо дробова частина дорівнює нулю, в іншому випадку *False.*

Клас *float* має обмеження на значення. Найбільше абсолютне значення цього класу становить *1.8e+308*, найменше абсолютне значення *2e-321*. При перевищенні верхнього граничного значення система оперує величиною *Inf* (Infinity) – нескінченність. При перевищенні нижнього граничного значення – нулем.

```
>>>fl1=1.7e306
>>>fl1
1.7e+306
>>> fl1*=1e1
>>> fl1
1.7000000000000001e+307
>>> fl1*=1e1
>>> fl1
1.7000000000000001e+308
>>>fl1*=1e1
>>>fl1
Inf.
```

Однак точність сприйняття значень класу *float* зникає при значеннях, значно менших за граничні:

```
>>> 2e100==2e100+1
True.
```

Оскільки значення класу *int* можуть приймати значення більші за верхню межу класу *float*, існує можливість генерування помилки:

```
>>> int1=2**300
>>> float(int1)
2.03703597633448 6e+90
```

```
>>> int1**=2
>>> float(int1)
4.149515568880993e+180
>>> int1**=2
>>> float(int1)
Traceback (most recent call last):
  File "<pyshell#36>", line 1, in <module>
    float(int1)
OverflowError: int too large to convert to float.
```

Комплексні значення є об’єктами класу *complex* і представляють двокомпонентне комплексне число, що складається з дійсної та уявної частин. Комплексне значення створюється виразом типу:

```
>>> cmp1=2+3j
>>> cmp1
(2+3j).
```

Комплексні значення підпорядковуються правилам алгебри комплексних чисел:

```
>>> cmp2=3+5j
>>> cmp3=5-2j
>>> cmp2, cmp3
((3+5j), (5-2j))
>>> cmp2+cmp3
(8+3j)
>>> cmp2-cmp3
(-2+7j)
>>> cmp2*3
(9+15j)
>>> cmp3**2
(21-20j)
```

```
>>> cmp3/5
(1-0.4j)
>>> cmp2*cmp3
(25+19j).
```

Властивості та методи класу *complex* обмежуються методом *conjugate* та дескрипторами *real* і *imag* з значеннями аналогічними класу *int.*

Клас *bool* передбачає існування всього двох значень:

True – істина

False – неправда

Присвоєння значень може відбуватися напряму з застосуванням зазначених літералів або через операції булевої алгебри:

```
>>> tbl=True
>>> fbl=False
>>> abl=(5>2)
>>> bbl=(5<2)
>>> tbl, abl, fbl, bbl
(True, True, False, False).
```

Докладніше правила булевої алгебри, формулювання та обробку булевих значень розглянуто в наступних практикумах.

Методи та властивості класу *bool* співпадають з відповідниками з класу *int*. При перетворенні типів необхідно враховувати, що будь-яке числове значення відмінне від нуля, непорожній рядок, або непорожній список є аналогами значення *True*, а числове значення рівне нулю, порожній рядок чи список – аналоги *False*.

1.2 Типи даних: послідовності або колекції

На відміну від скалярних типів даних, які розглянуто вище, *Python* допускає використання типів (класів), що є послідовностями або колекціями інших елементів. Найпростішими є рядки, кортежі та списки. Інші багатоелементні типи: словники, множини, тощо будуть розглянуті окремо в наступних практикумах.

Рядки – екземпляри класу *str* є послідовностями символів (або одно символьних рядків) і використовуються для вводу, зберігання, оброблення та виводу текстової інформації або іншої інформації приведеної до текстового вигляду.

Рядки створюються рядковими літералами – довільними послідовностями символів, обмежених зліва та справа знаками апострофа (’), або подвійних лапок ("). Іншим способом створення рядкового значення є перетворення в рядок числового чи іншого значення з застосуванням функції *str*:

```
>>> astr='This is string'
>>> bstr='This is another string'
>>> astr, bstr
('This is string', 'This is another string')
>>> type(astr)
<class 'str'>
>>> cstr=str(3.1415)
>>> dstr=str(5>4)
>>> cstr, dstr
('3.1415', 'True').
```

Рядки незмінювані – неможливо змінити частину рядка чи окремий символ, але можна створити новий рядок з елементів попереднього.

```
>>> dstr[1]='R'
Traceback (most recent call last):
  File "<pyshell#49>", line 1, in <module>
```

```
dstr[1]='R'
TypeError: 'str' object does not support item assignment
>>> dstr=dstr[0]+'R'+dstr[2:]
>>> dstr
'TRue'.
```

До кожного елемента рядка можна звернутися за його номером в рядку, починаючи з нуля, або одержати зріз[1] за позиціями в рядку:

```
>>> astr[0],astr[1],astr[2],astr[3],astr[4],astr[5],astr[6]
('T', 'h', 'i', 's', ' ', 'i', 's')
>>> bstr[8:15]
'another'.
```

Як усі класи типу колекції, *str* передбачає можливість використання функції *len* для визначення довжини послідовності – кількості одиничних об'єктів, що формують колекцію:

```
>>>len(astr), len(cstr)
(14, 6.
```

14 – довжина рядка *'This is string'* в символах, включаючи пропуски, 6 – довжина рядка *'3.1415'*.

Оператор *in* формує булеве значення відповідно до того входить вказаний підрядок (перший операнд) в рядок пошуку:

```
>>>'Th'in astr
True.
```

[1]Правила вибору індексів та побудови зрізів докладно описано для класу *list*. Для класів *str* та *tuple* застосовуються ті ж правила.

Клас *str* має більше сорока методів та властивостей, окремі з яких наведено нижче:

capitalize – першу літеру рядка переводить у верхній регістр (великі літери);

count – підраховує кількість включень заданого підрядка у рядок;

Find – повертає індекс першого включення підрядка в рядок;

Split – розбиває рядок на складові за заданим розділювачем;

Повний список методів класу *str* можна одержати як підказку середовища розробки, або з використанням функції середовища *help*.

Кортежі – екземпляри класу *tuple* – є незмінюваними колекціями різнорідних елементів. Складовими кортежу можуть бути ідентифікатори змінних або значення довільного типу, в тому числі і складних: рядків, списків, кортежів. Термін незмінювані означає неможливість змінити індивідуальне значення елемента кортежа. Проте з кортежа можна виділити індивідуальні чи групові складові і створити новий кортеж, значення якого можна присвоїти попередній змінній. Приклади кортежів:

```
>>> mytp=(1, 7, 3.2, True, 'String', 11)
>>> mytp[0], mytp[2], mytp[3]
(1, 3.2, True)
>>> newtp=mytp[:3], False, 'String_5', mytp[5]
>>> newtp
((1, 7, 3.2), False, 'String_5', 11)
>>> type(newtp)
<class 'tuple'>
>>> type(newtp[0])
<class 'tuple'>.
```

При побудові нового кортежа, його перша частина, одержана як зріз від 0 до 3 (не включаючи 3) перетворилася в кортеж. Тому для виділення окремих значень необхідно застосування подвійної індексації:

```
>>>newtp[0]
(1, 7, 3.2)
>>>newtp[0][2]
3.2.
```

Кортежі мають лише два методи: *count* та *index*, які працюють аналогічно методам класу *str*.

Списки – екземпляри класу *list* є змінюваними колекціями різнорідних елементів. З точки зору можливих елементів списки є повними аналогами кортежів. Суттєвою особливістю списків є їх здатність змінюватись: збільшувати або зменшувати кількість елементів, змінювати значення чи тип елементів. Списки створюються шляхом обмеження переліку елементів квадратними дужками:

```
>>> mylst=[1, 4, 6.2, 'List', 4.8]
>>> lst2=[5, mylst]
>>> mylst
[1, 4, 6.2, 'List', 4.8]
>>> lst2
[5, [1, 4, 6.2, 'List', 4.8]].
```

Елементи списку доступні для безпосередньої зміни, але при цьому список залишається на попередньому місці в пам’яті комп’ютера. Змінюються також похідні списки – ті, що включають змінений список як елемент (частково чи повністю).

```
>>> id(mylst)
805746850760
>>> mylst[0]=18
>>> mylst[3]=7.2
>>> mylst
[18, 4, 6.2, 7.2, 4.8]
>>> id(mylst)
```

```
805746850760
>>> lst2
[5, [18, 4, 6.2, 7.2, 4.8]].
```

Важливою властивістю списків (а також інших класів з послідовностями) є можливість одержання зрізів – часткова або повна передача значень в новий список чи в іншу адресу. Зріз реалізується через використання початкового та кінцевого індексів записаних через двокрапку. Перший індекс – перед двокрапкою – вказує номер першого елемента зрізу (нумерація починається з нуля). Якщо перший індекс опущено, він вважається рівним 0. Другий індекс – після двокрапки – вказує номер першого елемента, який не входитиме до зрізу. Якщо індекс опущений – вважається рівним кількості елементів в списку. Вказування неіснуючих індексів викликає помилку.

```
>>> lstid=[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]
>>> lstid
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
>>> lstid[0:16]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
>>> lstid[1:5]
[1, 2, 3, 4]
>>> lstid[:6]
[0, 1, 2, 3, 4, 5]
>>> lstid[5:]
[5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
>>> lstid[:]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
>>> lstid[5:5]
[]
>>> lstid[12:18]
[12, 13, 14, 15].
```

Індекси при генерації зрізів можуть приймати від'ємні значення, тоді справжнє значення індекса обчислюється як сума довжини списку та застосованого від'ємного значення:

```
>>>lstid[12:-2]
[12, 13]
>>>lstid[-7:]
[9, 10, 11, 12, 13, 14, 15]
>>>lstid[:-10]
[0, 1, 2, 3, 4, 5].
```

Клас *list* має досить велику кількість властивостей та методів:

append – додає значення в кінець списку;

clear – видаляє усі значення зі списку;

copy – створює копії списку;

count – рахує кількість включень вказаного елемента в списку;

extend – розширяє список за рахунок елементів ітерабельного об'єкта;

index – повертає індекс першого включення вказаного елемента;

insert – дозволяє вставити додатковий об'єкт в задану позицію списку;

pop – видаляє зі списку останній (вказаний) елемент і повертає його значення;

remove – видаляє зі списку перший елемент з заданим значенням;

reverse – переписує список в зворотному порядку;

sort – сортує список за вказаним правилом.

2 Завдання на комп'ютерний практикум

Дослідити властивості змінних розглянутих класів на прикладах завдань Додатку 2.

Скласти та відлагодити програму для демонстрації методів класу *list*.

3 Запитання для самоконтролю

1. Які форми запису цілочисельних значень існують в *Python* ?
2. Що означає термін “апроксимація числа з плаваючою крапкою” ?
3. Чи можна довільне цілочисельне значення перетворити в булеве, та чи можна його повернути назад ?
4. Скільки компонентів містить запис комплексного числа ?
5. Який тип має результат додавання комплексного числа та спряженого йому комплексного числа ?
6. Чи може значення класу *str* містити цифри ?
7. Чи можна значення класу *float* перетворити до класу *str* ?
8. Значення яких класів можуть бути елементами класу *tuple* ?
9. Список ml містить десять числових значень. Скільки значень буде містити:

```
n1=ml[:]
n2=ml[4]
n3=ml[5:5]
n4=ml[2:7]
n5=ml[:4]
n6=ml[3:-2]
```

10. Який формат має команда для вставки числа 8 на 3 місце списку *ml?*.

Література: [1] С. 122 – 126. [2] С. 36 –81. [3] С. 56 –69. [4].

Комп'ютерний практикум № 4

ВИКОРИСТАННЯ ЦИКЛІВ

Мета роботи: оволодіння теоретичною інформацією та здобуття практичних умінь використання операторів циклу.

1 Основні теоретичні відомості

1.1 Призначення та види циклів

Багаторазове циклічне повторення сукупності наперед визначених дій, заданих алгоритмів є одним з основних прийомів програмування. Саме за рахунок використання конструкцій циклів реалізуються практично усі наближені методи розрахунків, – розрахунки з матричними структурами, сортування послідовностей даних, тощо. Більшість мов програмування має стандартний набір операторів організації циклів.

Для випадків, коли кількість повторів програмного блоку відома до початку роботи циклу, а саме однотипне оброблення елементів послідовностей відомого розміру, сортування та матричні операції, застосовують цикл з лічильником (з параметром), який починається ключовим словом *for*. *Python* містить декілька способів створення циклів *for*, у тому числі скорочений внутрішньорядковий генератор послідовностей та зовнішній ітератор послідовностей.

Для випадків, коли кількість повторень блоку невідома, як для наближених обчислень до досягнення заданої точності, використовують цикли з передумовою або цикли з післяумовою. Цикли з передумовою виконуються якщо початкові умови відповідають правилу входу в цикл і до тих пір, доки ці умови не зміняться за рахунок дій всередині блоку тіла циклу. Цикли з післяумовою містять лише умову виходу з циклу, а значить тіло циклу виконується як мінімум один раз. *Python* містить лише оператор циклу з передумовою *while*, але він може бути модернізований до реалізації післяумови.

1.2 Цикли типу *for*

Організація циклів типу *for* в *Python* суттєво відрізняється від реалізації в таких традиційних мовах програмування як *Basic*, *Pascal* або *C*. В *Python* конструкція *for* є універсальним ітератором, що дозволяє послідовно видобувати значення (об'єкти) з колекцій. Загальний вигляд оператора циклу *for* може бути записаний як:

```
for <targ> in <object>:
    <operator>
    <operator>.
```

Ключовими елементами оператора є:

– слово *for* (для) – початок циклу;

– слово *in* (в)

– двокрапка (**:**) – вказує на початок тіла циклу, усі оператори якого вводяться зі стандартним відступом зліва.

Ключові елементи доповнюються параметрами циклу:

`<targ>` – ідентифікатор змінної, якому послідовно приписуються значення з послідовності;

`<object>` – послідовність елементів;

`<operator>` – оператори тіла цикла, що вводяться зі стандартним відступом зліва.

Послідовністю елементів може бути екземпляр довільного типу: рядок, кортеж, список та інші, які будуть розглянуті в наступних практикумах:

```
>>> mystring='Hello'
>>> for s in mystring:
    print(s)
H
e
l
```

```
l
o
>>> mylist=['Al', 'Fe', 'Cr', 'Ni', 'Ti']
>>> for metal in mylist:
    print(metal)
Al
Fe
Cr
Ni
Ti.
```

Якщо, цикл *for* необхідно виконати для створення або оброблення об'єктів-послідовностей (переважно списків) з наперед відомою кількістю елементів, то в якості елемента `<object>` може виступати генератор послідовностей `range()`, виклик якого має синтаксис:

range(start, stop, [step]) або ***range(stop),***

де ***start*** – початкове значення аргументу;

stop – значення аргументу, яке зупиняє генерацію; це значення відсутнє в послідовності;

step – значення кроку при переміщенні від ***start*** до ***stop***.

Якщо використовується формат ***range(stop),*** то значення інших аргументів приймаються за замовчуванням і дорівнюють відповідно: ***start***=*0*, ***step***=*1*. Якщо застосовується формат ***range(start, stop,[step])***, то аргумент ***step*** може бути пропущений і тоді приймається рівним одиниці.

Генератор ***range*** *()* (у версії *Python 3*) повертає не весь одержаний список відразу, а по одному значенню, що дозволяє уникати створення великої кількості додаткових списків і непродуктивного використання пам'яті комп'ютера.

Запис тіла циклу у вигляді:

```
for i in range(n):
    <operator i>….
```

є практично повним аналогом оператора циклу записаного мовою *Pascal*:

```
for i = 0 to (n-1) do
    <operator i>….
```

В обох випадках початкове значення індексу рівне нулю, крок рівний 1, останнє значення індексу *n-1*, де *n* звичайно має цілочисельне значення.

Наприклад фрагмент програми, наведений нижче формує два списки: зі значеннями індексів та їх квадратами:

```
ind_list=[] #створення порожнього списку індексів
sqw_list=[] #створення порожнього списку квадратів
for i in range(10):
    ind_list.append(i)
    sqw_list.append(i**2)
print(ind_list)
print(sqw_list).
```

Результат роботи програми має вигляд:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81].
```

Інші варіанти генератора *range* створюють наступні послідовності:

```
range(2, 6)      →   2, 3, 4, 5
range(1, 8, 3)   →   1, 4, 7.
```

Як тіло циклу може виступати довільна кількість окремих (простих) операторів, або складних операторів: інших циклів, умовних операторів, тощо. Прості оператори та заголовки складних операторів вводяться в програму з однаковим стандартним відступом зліва, а тіла складних операторів – з

подвійним[2] стандартним відступом зліва. Приклад застосування подвійного циклу показано нижче:

```
n,m=2,3
s=[] # порожній список сум індексів
p=[] # порожній список добутків індексів
for i in range(n):
    print('loop i=', i)
for j in range(m):
    print('loop j=',j)
    s.append(i+j)
    p.append(i*j)
print(s)
print(p).
```

Результат роботи програми:

```
loop i= 0
loop j= 0
loop j= 1
loop j= 2
loop i= 1
loop j= 0
loop j= 1
loop j= 2
[0, 1, 2, 1, 2, 3]
[0, 0, 0, 0, 1, 2].
```

Комбінація операторів *for* та *range* дозволяє створити однорядковий генератор послідовностей:

```
>>> mylst=[i**2 for i in range(3, 8, 2)]
```

[2] При написанні тексту програми зберігається однаковий стандартний відступ між заголовками тіла в складних операторах.

```
>>> mylst
[9, 25, 49].
```

або однорядковий оператор діалогового вводу значень списку заданого розміру:

```
z=[float(input('a['+str(i)+']=')) for i inrange(5)].
```

Приклад формування списку кортежів з трьох окремих списків **ax, bx, cx**:

```
ax=[0, 1, 2, 3, 4, 5, 6, 7]
bx=[0, 2, 6, 12, 20, 30, 42, 56]
cx=[-3, -1, 1, 3, 5, 7, 9, 11]
abcx=[]
for i in range(8):
    zx=tuple(x[i] for x in [ax, bx, cx])
    abcx.append(zx)
print(abcx).
```

Програма виводить інформацію:

```
[(0, 0, -3),(1, 2, -1),(2, 6, 1),(3, 12, 3),(4, 20,
5),(5, 30, 7), (6, 42, 9), (7, 56, 11)].
```

Такий прийом може бути дуже корисним для підготовки комплектів даних для виклику функцій. Замість окремих значень з різних списків (а значить змінюваних величин) передається кортеж значень (незмінюваний) з одного списку. У переважній більшості мов програмування програмний код такої функціональності буде займати значно більший об’єм.

1.3 Цикли типу *while*

Цикли з передумовою в *Python* реалізуються за рахунок оператора *while*, який має загальний синтаксис:

```
while <test1>:
    <operator 1>
    <operator 2>,
```

де **while** (доки) ключове слово початку циклу;

<**test1**> – вираз, результат якого є булеве значення *True* або *False*. Тіло циклу виконується тоді і тільки тоді коли результат випробування *True*;

<**operator 1**>,<**operator 2**> – набір простих чи складних операторів, що становлять тіло циклу і записуються зі стандартним відступом зліва. Умови виходу з циклу звичайно обчислюються всередині тіла циклу. Так наприклад фрагмент програми:

```
a=3
while a<6:
    print(a)
    a+=2
```

виводить результат:

```
3
5.
```

Оскільки *a=3 < 6,* то умова в операторі *while* повертає *True* і цикл починає виконуватись. Перший оператор тіла циклу виводить значення *a (3)*. Наступний оператор тіла циклу збільшує значення *a* на *2 (5)* і повертає програму на початок циклу.

Оскільки *a=5 < 6*, то умова в операторі *while* повертає *True* і цикл продовжує виконуватись. При наступній перевірці *a=7 > 6*, умова в операторі *while* повертає *False* і цикл припиняє роботу.

Однією з властивостей цикла з передумовою є те, що для початку його роботи необхідне виконання відповідної умови. На противагу цьому, в реальних програмах, виникає необхідність застосування алгоритму, коли перший раз тіло циклу виконується без умов, а тільки після цього перевіряється необхідність продовжувати чи зупиняти роботу.

Python не містить циклу з післяумовою, проте існує декілька способів модернізації циклу з передумовою. Один з них передбачає застосування індикаторної змінної, якій перед входом в цикл призначається значення, що задовольняє умову входу, а в самому тілі циклу – це значення модифікується відповідно до умов задачі. Ряд інших способів можуть опиратись на вирази булевої алгебри та ключове слово *break* – інструкція термінового переривання циклу. Більш докладно засоби булевої алгебри розглянуто в наступних практикумах.

2 Завдання на комп'ютерний практикум

Скласти та відлагодити набір програм (за завданням викладача), які використовують різні типи циклів: пошук суми або добутку числових рядів, обчислення з застосуванням списків, тощо. Зробити висновки. Для самоконтролю використовувати завдання Додатку 2.

3 Запитання для самоконтролю

1. В чому полягає різниця циклів з лічильником та циклів з передумовою?
2. Який тип циклу доцільно використовувати при роботі з елементами списку?
3. Скільки разів виконається тіло циклу, заданого кожним з наступних операторів?

– `for i in range(3):`

– `for i in range(1,3):`

– `for i in range(3,3):`

– `for i in range(2,5,3):`

– `for i in range(4,3):`

4. Для кожної з наступних задач обрати тип циклу, скласти та відлагодити програму:

а) задане натуральне число **N** та дійсне число **a**. Обчислити

$$P = \left(1 + \frac{1+a}{1^2}\right) \cdot \left(1 + \frac{2+a}{2^2}\right) \cdot \left(1 + \frac{3+a}{3^2}\right) \cdot \dots \cdot \left(1 + \frac{N+a}{N^2}\right)$$

б) задане натуральне число **N** та дійсне число **a**. Обчислити

$$S = \frac{1}{a} + \frac{2}{a^2} + \frac{3}{a^3} + \dots + \frac{N}{a^N}$$

Операцію степені (**) не застосовувати

в) задане натуральне число **N** та дійсне число **a**. Обчислити

$$S = (1 - a) + (2 + a^2) + (3 - a^3) + \dots + (N + (-1)^N \cdot a^N)$$

Операцію степені (**) не застосовувати

г) задані дійсні числа **a** та **b.** Обчислити найменше значення

$$P = (1 - a) \cdot (2 + a^2) \cdot (3 - a^3) \cdot \dots \cdot (N + (-1)^N \cdot a^N)$$

Яке більше або рівне b $(P \geq b)$. Операцію степені (**) не застосовувати.

Література: [1] С. 392 – 415. [2] С. 50 –51. [3] С. 23 –25, 56 – 69. [4].

Комп’ютерний практикум № 5

УМОВНИЙ ОПЕРАТОР. РОЗГАЛУЖЕННЯ ПРОГРАМИ

Мета роботи: оволодіння теоретичною інформацією та здобуття практичних умінь застосування умовних операторів для розгалуження програми.

1 Основні теоретичні відомості

1.1 Розгалуження програми

Розгалуження програми залежно від результату наперед заданих дій реалізує 3-й основний механізм програмування (поряд з лінійним виконанням операторів та циклічним повтором блоку операторів). Галуження алгоритму та програми, що його реалізує, може бути необхідним в математичних задачах, наприклад для реалізація різних розрахунків протягом розв’язання квадратного рівняння залежно від значення дискримінанту; в алгоритмах оброблення текстової інформації (збіг підрядка з шаблоном); в обробці даних – сортуванні, виборі, пошуку, тощо та у багатьох інших випадках.

Для організації розгалуження різні мови програмування застосовують набір способів, серед яких найпоширенішим є умовне галуження за допомогою оператора *if*.

1.2 Умовний оператор *if*

Схематично роботу умовного оператора показано на рисунку 5.1. Блок 1 містить умову. Якщо умова виконується – програма переходить до виконання блоку 2. Якщо умова блоку 1 не виконується – програма переходить до виконання блоку операторів 3. Після виконання блоку 2 або блоку 3 програма переходить до виконання спільного програмного блоку 4. Звичайно блок 3 та/або блок 4 не є обов’язковими і їх наявність залежить від місця умовного оператора в загальному алгоритмі

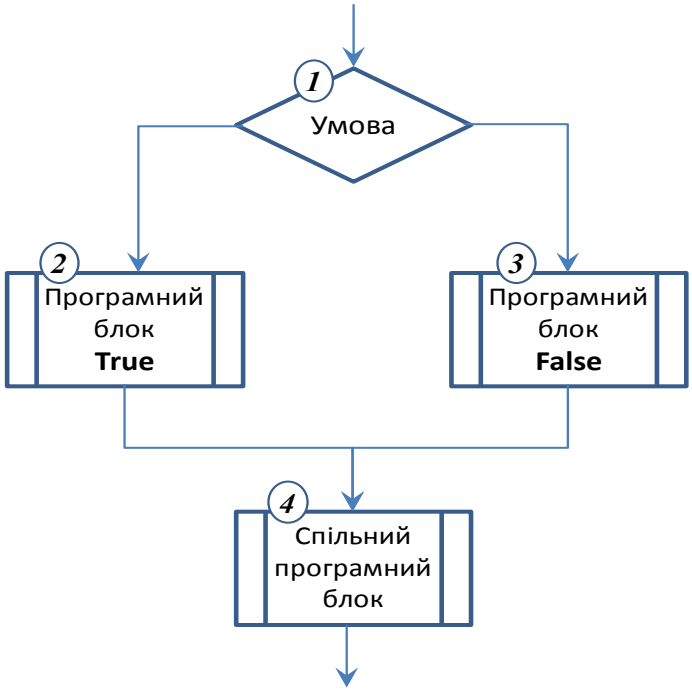


Рисунок 5.1 – Блок-схема умовного оператора

Оператор *if,* що реалізує наведену блок-схему має наступний синтаксис, який є одним з найбільш вживаних:

```
if <test>:
    <program block 2>
else:
    <program block 3>
<program block 4>,
```

де позначено:

if – ключове слово умовного оператора;

<test> – умова: константа, змінна або вираз, результат якого може бути розпізнаний як належний до класу *<class 'bool'>*. Після елементу *<test>* слідує обов'язковий символ – двокрапка (:) і наступний рядок починається з обов'язкового стандартного відступу зліва;

<program block 2> – блок операторів, які виконуються, якщо результат *<test>=True*;

else – ключове слово другої частини умовного оператора – керує виконанням дій у випадку, якщо *<test>=False*. Закінчується двокрапкою, а наступний рядок починається з обов'язкового стандартного відступу зліва;

<program block 3> – блок операторів, які виконуються, якщо результат *<test>=False*;

<program block 4> – блок операторів (продовження програми) які виконуються після завершення роботи умовного оператора, незалежно від виконання чи невиконання умови.

Наведений синтаксис запису умовного оператора є найпоширенішим, але не єдино можливим. В програмах використовують як скорочені, так і розширені форми оператора.

Скорочена форма виду:

```
if <test>:
    <programblock 2>
<programblock 4>
```

відрізняється від повної тим, що не включає у свою структуру ключове слово *else* та, відповідно, не виконує ніяких специфічних дій у випадку, коли *<test>=False.* Якщо результат *<test>=True*, виконуються дії *<programblock 2>*, після чого програма виходить з умовного оператора і виконується *<programblock 4>*. Якщо результат *<test>=False*, то програма одразу виходить з умовного оператора і виконується *<programblock 4>*.

У випадку, коли умова складається з одного оператора, скорочена форма умовного оператора може бути записана в один рядок:

```
if <test>:<programblock 2>.
```

Тоді програма продовжується з наступного рідка без відступу. Таким одним оператором може бути двомісний (або більше) оператор присвоювання. Наприклад наведений рядок програми обмінює значеннями змінні *a* та *b*, у випадку якщо *a* > *b*:

```
if a>b: a,b=b, a.
```

Багатомісний (розгалужений) оператор *if* має загальний синтаксис виду:

```
if <test_1>:
    <program block 1>
elif <test_2>:
    <program block 2>
…
elif <test_n>:
    <program block n>
else:
    <programblockk>.
```

Приклад застосування багатомісного оператора if наведено в програмі:

```
#if program 2
x=[i for i in range(10)]
for i in range(10):
    if i<3:
        print('< 3')
    elif i==4:
        print('= 4')
    elif i<7:
        print('= 5..6')
    else:
        print('= 3, 7..9')
print('end of program').
```

Результати роботи програми:

```
<= 3
<= 3
<= 3
= 3, 7..9
= 4
```

```
= 5..6
= 5..6
= 3, 7..9
= 3, 7..9
= 3, 7..9
end of program.
```

1.3 Генерація булевих значень та дії з ними

Розглянутий вище оператор галуження *if*, оператор циклу *while* (комп'ютерний практикум № 4) та деякі інші конструкції мови програмування *Python* опираються на булеві значення – *True* або *False*. У ряді прикладів обмежуються одержанням булевих значень безпосередньо в заголовку оператора, який їх використовує і переважно через операції порівняння. Однак в мові *Python* існує декілька способів одержання відповідних значень.

1. Безпосереднє присвоєння значень:

a = True,

b = False.

Таким способом створюються статичні булеві змінні з досить вузькими можливостями застосування. Однак вони можуть бути корисними при відлагоджуванні програм.

2. Застосуванням операцій порівняння. У програмах мовою *Python* застосовуються наступні операції порівняння:

>	більше	*a > b*	*a* більше b
<	менше	*a < b*	*a* менше b
>=	більше або дорівнює	*a >= b*	
<=	менше або дорівнює	*a <= b*	
==	дорівнює	*a == b*	
!=	не дорівнює	*a != b*	
is	є (ідентично)	*a is b*	

not is не ідентично *a is not b*

Приклади застосування:

```
>>> a, b, c = 3, 4, 5
>>> d=5
>>> f, g = 4., 5.
>>> a>b  # a=3, b=4
False
>>> a<=b # a=3, b=4
True
>>> a==b # a=3, b=4
False
>>> b!=c # b=4, c=5
True
>>> c==d # c=5, d=5
True
>>> c is d   # c=5, d=5
True
>>> c==g # c=5, g=5.
True
>>> c is g   # c=5, g=5.
False
>>> k=a<b
>>> k
True.
```

Операції порівняння можуть бути зібрані у багатомісні вирази:

```
>>> a<c>b    # a=3, b=4, c=5.
True
>>> c==d==g  # c=5, d=5, g=5.
True
```

```
>>> m=5
>>> c is d is m  # c=5, d=5, m=5.
True
>>> c==d>b>a<g        # c=5, d=5, b=4, a=3, g=5.
True
```

3. Застосуванням ключового слова *in* для визначення, чи входить елемент в значення одного з впорядкованих типів.

```
>>> alist=[4, 5, 6, 8, 9]
>>> 4 in alist
True
>>> 6 in alist
True
>>> 7 in alist
False
>>> btuple=(11, 12, 13)
>>> 10 in btuple
False
>>> 12 in btuple
True
>>> cstr='AaBbCcDdEeFf'
>>> 'f' in cstr
True
>>> 'g' in cstr
False
>>> '1' in cstr
False
>>> 'aBb' in cstr
True
>>> str_list=['red', 'green', 'blue']
>>>'red' in str_list
```

```
True
>>>'black' in str_list
False.
```

4. Застосуванням значення довільного типу в якості булевого значення. Так, наприклад, будь-яке ненульове числове значення прирівнюється до *True*, а нульове – до *False*. Непорожній об'єкт впорядкованого типу – *True*, порожній – *False.*

```
>>> al=[]     # Порожній список
>>> bl=[1,2] # Непорожній список
>>> if al:
    print('True')
>>> if bl:
    print('True')
True
>>> astr=''  # Порожній список
>>> bstr='String'     # Непорожній список
>>> if astr:
    print('True')
>>> if bstr:
    print('True')
True.
```

Булеві значення можуть об'єднуватись у вирази знаками (літералами) булевих операцій: *and, or, not.*

Операція *and* – логічне (булеве) множення. Результатом операції є *True*, тільки у випадку, коли кожен з елементів, об'єднаних операцією має значення *True.* В інших випадках результат операції – *False*.

Операція *or* – логічне (булеве) додавання. Результатом операції є *True*, якщо один, або обидва елементи, об'єднані операцією мають значення *True*. Якщо обидва елементи мають значення *False* то результат операції *False*.

Операція *not* – одномісне логічне заперечення. Результатом операції є значення, протележне значенню елемента, перед яким застосовано операцію *not*. Дія операції розповсюджується лише на елемент, безпосередньо перед яким знаходиться літерал операції. Якщо логічне заперечення необхідно застосувати до результату іншої логічної операції – необхідно застосовувати круглі дужки:

```
>>> a, b, c =True, True, False
>>> a and b
True
>>> a and c
False
>>> a and not c
True
>>> a or c
True
>>> a or b
True
>>> not a and c
False
>>> not c and a
True
>>> not (a and c)
True
>>> not (c and a)
True.
```

2 Завдання на комп'ютерний практикум

Скласти та відлагодити набір програм (за завданням викладача), які використовують умовні оператори для розгалуження програми: сортування списків, аналіз властивостей наборів чисел, тощо. Зробити висновки.

3 Запитання для самоконтролю

1. Чи можна записати умовний оператор одним рядком, і якщо так, то який синтаксис такого запису?
2. Як розгалузити хід виконання програми на три потоки (наприклад за значеннями параметра: =0, >0, <0)?
3. Чи може умовний оператор знаходитись всередині програмного блоку іншого умовного оператора?
4. Що спільного в умовного оператора *if* та оператора циклу з передумовою *while*?
5. Існують дві змінні з булевими значеннями. Записати вираз, що імітує роботу операції XOR, відомої в ряді мов програмування: операція повертає *True* тоді, коли один, і тільки один з елементів має значення *True*.

```
def xor(x,y):
    return (not(x and y)and(x or y)).
```

Література: [1] С. 376 – 390. [2] С. 47 – 50. [3] С. 56 – 69.[4].

Комп'ютерний практикум № 6

ФУНКЦІЇ КОРИСТУВАЧА. МОДУЛЬ RANDOM

Мета роботи: оволодіння принципами створення та застосування функцій в середовищі *Python* та здобуття практичних навичок роботи з функціями. Оволодіння принципами застосування основних елементів модулів *random* та *time*.

1 Основні теоретичні відомості

1.1 Функції користувача

Усі сучасні системи програмування містять засоби створення підпрограм – частково ізольованих програмних блоків, що можуть виконувати задані дії. Застосування підпрограм дозволяє структурувати програму, спростити відлагоджування її окремих блоків, добитися більшої прозорості та спростити читання коду програми.

В програмах мовою *Python* роль підпрограми відіграє програмний блок – функція, створена інструкцією *def* за загальним синтаксисом:

```
def <f_name>([<param list>]):
    ['''
    Short comment about what function does
    '''']
    <program block>
    [return <value>],
```

де ***def*** – ключове слово створення функції оператора;

<f_name> – ім'я функції, яка створюється.

Ім'я функції має бути лаконічним, мати відношення до дій, які виконує функція, відповідати вимогам до ідентифікаторів *Python* та не порушувати цілісність простору імен;

([<param list>]) – список параметрів функції. Список може бути порожнім, тоді за іменем функції слідують порожні круглі дужки: *my_fn1()*, або може включати одне або декілька імен змінних або значень: *my_fn2(x, y)*. Кожна змінна, вказана у списку параметрів може бути довільного типу. Елемент ***([<param list>])*** завершується двокрапкою (:) і наступний рядок починається з обов'язкового стандартного відступу зліва;

```
['''
Short comment about what function does
'''],
```

необов'язковий, але рекомендований рядок коментарів, в якому коротко викладено призначення функції, перелік і тип вхідних параметрів, формат результату. Такий коментар полегшує читання та розуміння коду функції. Також текст коментаря виводиться у відповідь на запит *help(<f_name>)*.

<program block> – набір операторів, що утворюють тіло функції;

[return <value>] – рядок повернення значення в програму, з якої було викликано функцію. Значення ***<value>*** може бути довільного типу.

Оператор опису функції ***def*** може бути використаний у будь-якому місці програми. За цим правилом *Python* суттєво відрізняється від ряду інших мов програмування. Єдина вимога – функція має бути описана до свого першого використання. Однак, «правила гарного тону» при програмуванні рекомендують розміщувати усі описи функцій на початку тексту програми до початку її виконання; уникати опису функцій всередині інших функцій, циклів, умовних операторів. Як просту, розглянемо функцію одержання оберненого числа. (обернене до числа x записується як $1/x$, або як x^{-1}).

```
def my_inverse(x):
    if x==0:
        return(None)
    else:
        return(1./x).
```

Як показано, тіло функції містить умовний оператор, який розгалужує функцію на дві гілки, кожна з яких завершується власним оператором *return*. Перша гілка: якщо аргумент рівний 0, і для нього обернене число не визначено, то функція повертає *None* – спеціально зарезервований в *Python* об'єкт, який не має ні значення ні типу (точніше тип *NoneType*). Друга гілка: "в іншому випадку", якщо аргумент не рівний 0, функція повертає значення оберненого числа для вказаного аргументу.

Описана таким чином функція може бути використана в інтерактивній оболонці:

```
>>> my_inverse(3)+ my_inverse(2)
0.8333333333333333.
```

або в програмі: у правій частині операторів присвоювання, чи як аргумент функції:

```
z=my_inverse(4)+my_inverse(2)
print(my_inverse(z))
print(my_inverse(my_inverse(5)))
1.3333333333333333
5.0.
```

Основні переваги використання функцій:

- функції дозволяють створювати програмний код один раз;
- функції дозволяють заховати непотрібну складність від користувача;
- функції роблять код простішим для читання та розуміння;
- функції дозволяють організувати програму логічно.

1.2 Формування списку параметрів

У випадку, коли кількість вхідних параметрів функції перевищує 2-3, можуть виникнути складнощі з дотриманням порядку їх слідування в рядку виклику функції. Крім того, у випадках значної кількості параметрів, частина з

них достатньо рідко змінює своє значення, тому для них може бути задане значення за замовчуванням, і вказувати його значення в рядку виклику функції не є обов'язковим.

Наприклад, опис функції зваженого середнього може включати величини (a, b, c) та їх вагові коефіцієнти (aw, bw, cw) і мати вигляд:

```
def weit_avg(a, b, c, aw=0.5, bw=0.3, cw=0.2):
    print('a=', a, 'aw=', aw)
    print('b=', b, 'bw=', bw)
    print('c=', c, 'cw=', cw)
    avg=(a*aw+b*bw+c*cw)/(aw+bw+cw)
    return(avg).
```

Різні варіанти виклику функції з *Python Shell*:

```
>>> weit_avg(5, 7, 9, 0.3, 0.3, 0.3)
a= 5 aw= 0.3
b= 7 bw= 0.3
c= 9 cw= 0.3
7.000000000000001
>>> weit_avg(5, 7, 9)
a= 5 aw= 0.5
b= 7 bw= 0.3
c= 9 cw= 0.2
6.3999999999999995
>>> weit_avg(5, 7, 9, bw=0.7)
a= 5 aw= 0.5
b= 7 bw= 0.7
c= 9 cw= 0.2
6.571428571428571.
```

При виклику функції діють правила:

– порядок слідування параметрів у рядку виклику функції має повністю відповідати порядку вказаному в заголовку опису функції;

– параметри, яким в описі функції надано значення "за замовчуванням", можуть бути пропущені у рядку виклику функції;

– якщо в рядку виклику пропущено значення частини параметрів, то значення для параметрів, що слідують за пропущеними передаються у повнорозмірному форматі: *prm=<value>*.

Крім функцій з чітко заданою кількістю вхідних параметрів використовуються функції зі змінною кількістю параметрів. Формат заголовка функції має вигляд:

def <f_name>(prm1, prm2, …, *args, **kwargs).

При виклику функції **args* може бути замінений довільною кількістю значень, які всередині функції формують кортеж *args*. Параметр ***kwargs* – якщо присутній в описі замінюється довільною кількістю пар *prm=<value>* і всередині функції перетворюється на словник *kwargs*[3].

1.3 Повернення значень

Програмний код функції може не містити оператора *return*, містити один такий оператор чи навіть декілька. Якщо оператор *return* відсутній, то така функція не повертає в програму ніяких значень і єдиним способом прояву власної роботи може бути наявний у функції оператор виводу.

Якщо оператор(и) *return* присутні, то робота функції завершується першим з них. Залишок програмного коду ігнорується. Значення розміщене в операторі *return* передається в основну програму і може бути присвоєне змінній або опрацьоване в інший спосіб. Таким чином, функція може повернути в програму лише одне значення. Якщо існує потреба повернути в програму більше одного значення, наприклад три координати точки в тривимірному просторі, їх можна об’єднати в значенні одного з типів-послідовностей: списках *(list)* або кортежах

[3]Структуру класу словник (*dictionary)* буде розглянуто в роботах другого кредитного модуля.

(tuple). В таких випадках значення необхідно присвоїти змінній, а потім видобути його компоненти.

Наприклад наведена нижче функція обчислює за координатами в полярній системі (*ro* – радіус-вектор, *fi* – кут, радіан) координати в декартовій системі (x, y) і повертає їх у вигляді кортежу:

```
def polar_cartes(ro, fi):
    from math import sin, cos
    xx=ro*cos(fi)
    yy=ro*sin(fi)
    return((xx,yy)).
```

В програмі така функція може бути використана, як:

```
coord=polar_cartes(2.4, 0.85)
x_c=coord[0]
y_c=coord[1]
print('X=', x_c, '  Y=', y_c).
```

Що призведе до виводу:

```
X= 1.5839595501239572 Y= 1.8030729723367025.
```

Схожу технологію застосовано в прикладі (варіант 2.) комп'ютерного практикуму № 1, для передачі в програму коренів квадратного рівняння.

1.4 Простір імен

Функції *Python* є прикладом елемента (поряд з класами), які створюють власний простір імен. Розуміння простору імен є ключовим для успішного створення та застосування функцій.

Якщо не вказано інше, при виклику функцій для кожної з них формується свій власний простір імен, що означає:

– імена визначені всередині опису функції видимі (можуть використовуватись) тільки програмному коду всередині функції. До цих імен не

можна звернутися ззовні ні до початку роботи функції, ні після завершення роботи;

– імена визначені всередині функції не вступають в конфлікт з іменами, що знаходяться за межами функції.

Відносно окремо взятої функції існує 4 простори імен: локальний, охоплюючий, глобальний та вбудований. Їх співвідношення показано на рисунку 6.1.

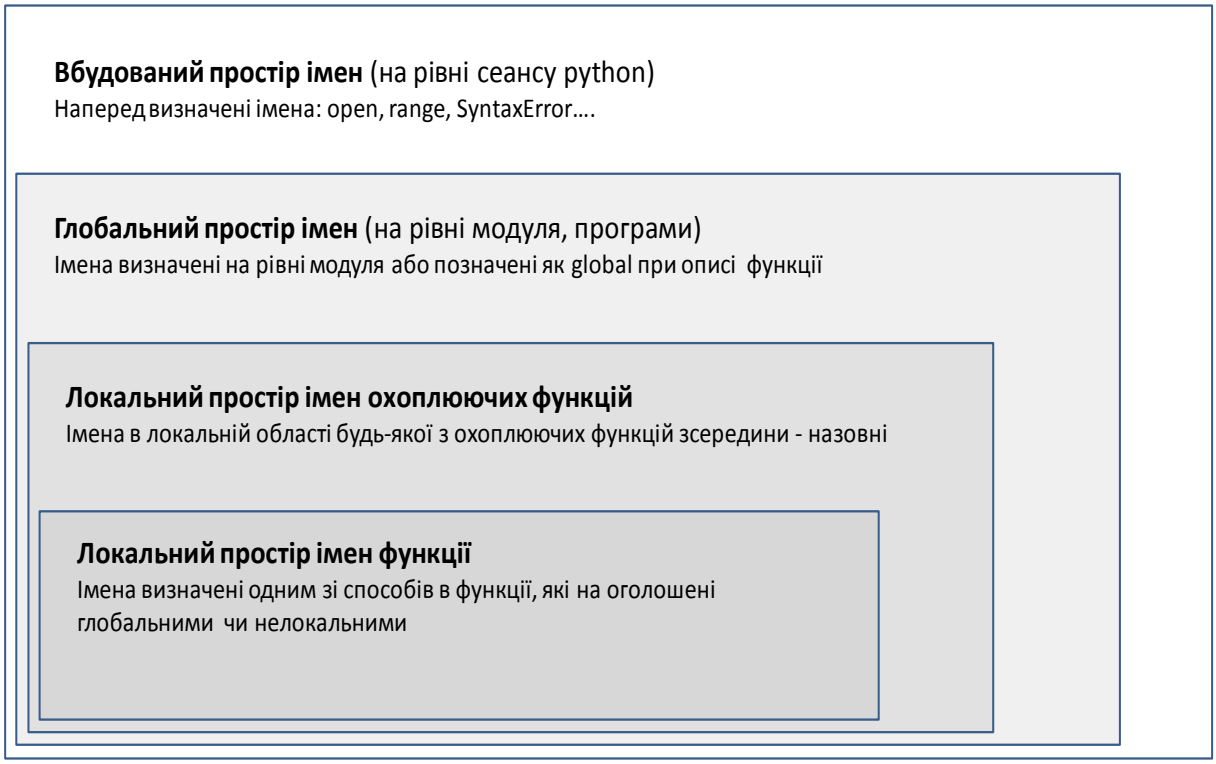


Рисунок 6.1 – Вкладеність просторів імен в *Python*

Модулі (програми) створюють глобальні простори імен, функції – локальні простори імен. Ці області пов’язані між собою наступним чином: охоплюючий модуль – глобальна область видимості. Кожен модуль – це глобальна область видимості – простір імен, в якому створюються змінні верхнього рівня. Глобальні змінні ззовні модуля стають атрибутами об’єкта, а всередині – використовуються як звичайні змінні.

Глобальна область видимості охоплює єдиний файл (програми, модуля).

Кожен виклик функції створює нову локальну область видимості. Виклик функції самою себе (рекурсивний) теж створює нову локальну область видимості.

Операції присвоювання створюють локальні імена, якщо їх не оголошено глобальними чи нелокальними. Усі імена, які створюються (яким присвоюються значення) всередині функції, а також імена параметрів виклику функції (перераховані в заголовку опису) вважаються локальними. Їх значення недоступні ззовні (крім передачі значень параметрам при виклику функції) і з їх допомогою неможливо передати значення з функції назовні.

Якщо всередині функції необхідно присвоїти значення змінній створеній на рівні модуля (глобальній) то відповідне ім'я в описі функції оголошується глобальним за допомогою інструкції *global*. Якщо ім'я змінної являється локальним для охоплюючої функції, то для присвоєння йому значення його необхідно оголосити нелокальним з використанням інструкції *nonlocal*.

Розглянемо приклад:

```
# test of name spaces
def myf1(x): #x - local for myf1
    print('myf1 start with prm=' ,x)
    z1=4 # local for myf1
    print(xg,yg,z1+x)
    print('myf1 finish')
def myf2(s): #s  - local for myf2
    print('myf2 start with prm=' ,s)
    global xg
    xg=25
    def myf3(d): # d - local for myf3
        print('myf3 start with prm=' ,d)
        nonlocal z2
        z3=z2+d
        print(z3)
        z2=36
        print('myf3 finish')
    z2=6 # local for myf2
```

```
    print(xg,yg,z2+2*s)
    myf3(s+z2)
    print('myf1 finish')

xg,yg,zg=5, 10, 15 # global value
print(xg, yg, zg)
myf1(zg)
myf2(zg)
print(xg, yg, zg).
```

В результаті виконання програми одержимо вивід:

```
5 10 15
myf1 start with prm= 15
5 10 19
myf1 finish
myf2 start with prm= 15
25 10 36
myf3 start with prm= 21
27
myf3 finish
myf1 finish
25 10 15.
```

Іншими словами: ім’я, яке використовується всередині функції має бути ініційоване або всередині функції (наприклад *z3* у функції *myf3*), або у одній з охоплюючих функцій (*z2* у функції *myf3* ініційована в *myf2*), або в тексті основної програми (*xg*, *yg* використовується у різних функціях). Якщо змінні, описані на різних рівнях мають одне ім’я, то доступною залишається тільки "найлокальніша з них". Для того, щоб мати можливість змінювати значення нелокальних та

локальних змінних всередині функції, перед першим їх використанням необхідно оголосити їх відповідним чином.

Правила "гарного тону" в програмуванні рекомендують максимально уникати використання нелокальних імен і тим більше уникати випадків зміни значення нелокальної змінної оскільки це суттєво погіршує прозорість програми та можливість її відлагоджування.

1.5 Рекурсивні функції

Як і значна кількість мов програмування *Python* підтримує використання рекурсивних функцій – функцій, які можуть викликати самі себе. Рекурсія – достатньо корисний прийом в програмуванні, який в окремих випадках може бути альтернативою циклам з умовою, хоч не завжди ефективною.

Розглянемо функцію обчислення факторіалу *n!*. Пригадаємо, що функція факторіал існує для цілочисельних аргументів більших або рівних нулю. Причому:

```
0! = 1
1! = 1
2! = 1*2=2
3! = 1*2*3=6
....
```

Функція має вигляд:

```
def fctr(n):
    if n==0 or n==1:
        return(1)
    else:
        return(n*fctr(n-1)).
```

Порядок роботи функції наступний: якщо значення аргументу дорівнює 0 або 1, функція повертає добуток аргументу на значення функції від аргументу зменшеного на 1.

В інших випадках рекурсивна функція може викликати більшу кількість своїх копій. При реалізації рекурсивних функцій необхідно ретельно відслідковувати зміну аргументу функції при її виклику.

1.6 Модуль *random*

Серед задач, які розв'язуються комп'ютерними програмами досить часто зустрічаються такі, що пов'язані з генеруванням і використанням послідовностей випадкових чисел. Наприклад, сукупність чисельних методів Монте-Карло, що використовуються для інтергування функцій, методів комп'ютерного імітаційного моделювання, тощо.

Для генерування послідовностей випадкових чисел в *Python* використовується модуль *random*, який може бути імпортований в програму стандартним способом:

import random.

Насправді, генерація дійсно випадкових чисел є досить складним завданням, тому програмним методом генеруються так звані "псевдовипадкові" числа. Крім того, функції модуля *random* дозволяють одержувати "псевдовипадкові" об'єкти різних типів. Нижче наведено список основних функцій модуля *random* (табл. 6.1).

Таблиця 6.1 – Список функцій і констант модуля *random*

Функція	**Значення, аргументи**
1	2
random.seed([X],version=2)	Ініціювання генератора випакових чисел визначеним або унікальним ключем. Якщо аргумент не вказаний – використовується поточний час.
random.getstate()	Повертає внутрішній стан генератора
random.setstate(state)	Відновлює внутрішній стан генератора за значенням *state*, одержаним функцією *getstate*
random.randrange(start, stop, step)	Повертає випадкове число з послідовності заданої значеннями *start, stop, step*
random.randint(A,B)	Повертає випадкове ціле число N, таке, що $A \le N \le B$
random.choice(seq)	Повертає випадковий елемент непорожньої послідовності *seq*

Продовження таблиці 6.1

1	2
random.shuffle(seq,[rand])	Перемішує елементи послідовності*seq*. Працює лише зі змінюваними об'єктами.
random.sample(pop, k)	Повертає список довжиною *k* з послідовності *pop*
random.random()	Повертає випадкове число в межах від 0 до 1
random.uniform(A, B)	Повертає випадкове число T з плаваючою крапкою, таке, що $A \leq T \leq B$
random.triangular(low, high, mode)	Повертає випадкове число T з плаваючою крапкою, таке, що $low \leq T \leq high$ іпідпорядковується трикутному розподілу з модою *mode*
random.betavariate(alpha, beta)	Повертає значення між 0 та 1, що підпорядковується одномодальному бета-розподілу. *alpha>0*, *beta>0*
random.expovariate(lambd)	Повертає значення від 0 до +∞, якщо *lambd>0* і від 0 до -∞, якщо *lambd<0.* Значення підпорядковується експоненційному розподілу. lambd = 1/(середнє бажане) ,≠0
random.gammavariate(alpha, beta)	Значення, що відповідає гаммарозподілу. *alpha>0*, *beta>0*
random.gauss(val, stdev)	Значення, що відповідає розподілу Гауса, з середнім *val* та дисперсією *stdev*
random.normalvariate(val, stdev)	Значення, що відповідає нормальному розподілу, з середнім *val* та дисперсією *stdev*
random.gammavariate(alpha, beta)	Значення, що відповідає гаммарозподілу. *alpha>0*, *beta>0*
random.gauss(val, stdev)	Значення, що відповідає розподілу Гауса, з середнім *val* та дисперсією *stdev*
random.normalvariate(val, stdev)	Значення, що відповідає нормальному розподілу, з середнім *val* та дисперсією *stdev*
random.gammavariate(alpha, beta)	Значення, що відповідає гаммарозподілу. *alpha>0*, *beta>0*

2 Завдання на комп'ютерний практикум

Скласти та відлагодити набір програм (за завданням викладача), які використовують функції користувача. Зробити висновки.

3 Запитання для самоконтролю

1. Які переваги використання функцій?
2. Скільки значень функція може повернути в точку виклику оператором *return*?
3. Задано фрагмент програми:

```
def my1():
    x=5
    print(x)
def my2(z):
    print(z)
def my3(z):
    global x
    x+=z
    print(x)
…
x=8
my1()
my3(5)
my2(x+3)
my3(3)
print(x)
….
```

Які значення послідовно будуть виведені програмою (5 значень)?

4. Що таке рекурсивні функції і для чого вони використовуються?

5. Запишіть функцію, яка буде повертати значення кореня лінійного рівняння вигляду: $a \cdot x + b = 0$ за вказаними величинами a та b.

6. Запишіть функцію, яка буде повертати інформацію про те чи співпадають значення двох її аргументів.

Література: [1] С. 461 – 502. [2] С. 168 – 185. [3] С. 56 – 69, 152 – 174. [4].

Комп'ютерний практикум № 7

ЗАСТОСУВАННЯ МОДУЛЯ NUMPY

Мета роботи: оволодіння основами застосування структур та функцій модуля *numpy*. Засвоєння методики роботи з багатовимірними масивами даних – створення, модифікування, дослідження властивостей.

1 Основні теоретичні відомості

1.1 Модуль *numpy*

Для повноцінного використання мови *Python* в якості платформи проведення математичних обчислень базова версія непридатна з декількох причин, основними з яких можуть бути названі відсутність підтримки багатовимірних масивів з типізацією даних та повільна робота математичних алгоритмів оброблення послідовностей, пов'язана з особливостями мови, як інтерпретатора. Для виправлення цих недоліків, починаючи з 1995 року розроблялись зовнішні бібліотеки *Numeric*, *NumArray*. Починаючи з 2005 року розробки об'єднанні з метою створення бібліотеки *numpy*.

Numpy – відкрита бібліотека, що змістовно належить до стеку модулів *SciPy* і реалізує ряд можливостей, серед яких:

– потужні об'єкти для створення n-вимірних масивів даних;

– засоби інтегрування коду на *C/C++* та *Fortran*

– значна кількість функцій для роботи з матричними структурами, функцій лінійної алгебри, перетворення Фур'є, тощо.

Суттєвою перевагою застосування функцій *numpy* є те, що вони реалізовані компілюючою мовою програмування і забезпечують швидкість виконання властиву програмам на *C/C++*.

1.2 Встановлення модуля *numpy*

Для використання засобів модуля *numpy* його спочатку необхідно встановити. Встановлення модуля можливо декількома способами залежно від операційної системи та інших особливостей комп'ютера.

Numpy є частиною більшості пакетів, розглянутих в практикумі № 1 (*Anaconda*, *PythonXY*, тощо). Файли для установки *numpy* можуть бути безкоштовно одержані з офіційного сайту розробника https://www.scipy.org/scipylib/download.html. Найбільш надійний спосіб встановлення модуля засобами штатного менеджера пакетів (*pip – Pip Installs Python*). Встановлення модуля таким способом вимагає підключення до мережі Internet та прав адміністратора в режимі командного рядка. Встановлення здійснюється однією командою:

```
$> pip install numpy.
```

Залежно від виду уже встановленого забезпечення, його версій, система може запропонувати інший формат команд, який включає оновлення самого менеджера пакетів – pip. Після встановлення модуля його структури та функцій стають доступними через операцію імпорту:

```
import numpy.
```

1.3 Багатовимірні масиви

Значна кількість технічних розрахунків опирається на матриці та їх властивості. Матриці – це двовимірні масиви даних, створення яких засобами базової версії *Python* неефективне і практично неможливе. Модуль *numpy* розширює можливості базової версії створюючи структуру *numpy.ndarray* – багатовимірний гомогенний масив. Це таблиця елементів одного типу, які індексуються набором цілих чисел, кількість яких (індексів) дорівнює кількості вимірів. Кількість вимірів або кількість осей називається рангом масиву (не плутати з рангом матриці).

Для початку роботи з масивами модуля *numpy*, модуль необхідно імпортувати. Рекомендується при імпорті надавати модулю скорочену назву – *пр:*

```
import numpy as np.
```

Масив *numpy.ndarray* може бути створений декількома способами, серед яких:

– перетворення списку або кортежу;

– зміна форми списку, кортежу чи існуючого масиву;

– безпосереднє створення заданої структури з вказаними даними (більше підходить для діалогового режиму, або створення даних для відлагоджування);

– застосування функцій генераторів: *ones, zeros, empty, eye,* тощо.

```
>>> a=[1,2,3,6,7,8] # list
>>> b=(3,6,9,12) # tuple
>>> print(type(a), type(b))
<class 'list'><class 'tuple'>
>>> an=np.array(a)
>>> bn=np.array(b)
>>> print(type(an), type(bn))
<class 'numpy.ndarray'><class 'numpy.ndarray'>
>>> an2=np.array(a).reshape(2,3)# 2x3 array
>>> an
array([1, 2, 3, 6, 7, 8])
>>> bn
array([ 3,  6,  9, 12])
>>> an2
array([[1, 2, 3],
       [6, 7, 8]])

>>> cx=np.array([[1.2, 4,6], [3,2,1], [4,8,3]])
>>> cx
array([[ 1.2,  4. ,  6. ],
```

```
       [ 3. ,  2. ,  1. ],
       [ 4. ,  8. ,  3. ]])
>>> dx=np.ones((3,4))
>>> dx
array([[ 1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.]])
>>> dz=np.zeros((2,3,2))
>>> dz
array([[[ 0.,  0.],
        [ 0.,  0.],
        [ 0.,  0.]],

       [[ 0.,  0.],
        [ 0.,  0.],
        [ 0.,  0.]]]).
```

Коли в програмі необхідно створити матрицю заданого розміру і заповнити її значеннями в діалоговому режимі, можна використати наступний програмний фрагмент (за умови що модуль *numpy* імпортований як *np*):

```
n_r=3 # number of raws
n_c=4 # number of columns
av=np.empty((n_r,n_c)) # empty matrix
for i in range(n_r):
    for j in range(n_c):
        av[i,j]=input('av[  '  +str(i)+',  '+str(j)+'
    ]=').
```

Повний перелік функцій-генераторів масивів *numpy.array* наведено в таблиці 7.1.

Таблиця 7.1 – Перелік функцій-генераторів масивів *numpy.array*

Назва функції	Аргументи	Призначення
empty	*(shape [,dtype, order])*	Масив заданого розміру, з заданим типом елементів, без значень
empty_like	*(a[,dtype, order, subok])*	Масив з розміром та типом як a, без значень
eye	*(n[,m,k,dtype])*	Двовимірна матриця розміру n з “1” по головній діагоналі, решта “0”
identity	*(n[, dtype])*	Одинична матриця заданого розміру
ones	*(shape[,dtype, order])*	Масив заданого розміру, з заданим типом, заповнений “1”
ones_like	*(a[,dtype, order, subok])*	Масив з розміром та типом як a, заповнений “1”
zeros	*(shape[,dtype, order])*	Масив заданого розміру, з заданим типом, заповнений “0”
zeros_like	*(a[,dtype, order, subok])*	Масив з розміром та типом як a, заповнений “0”
full	*(shape, fill [,dtype, order])*	Масив заданого розміру, з заданим типом, заповнений “fill”
full_like	*(a,fill[,dtype, order, subok])*	Масив з розміром та типом як a, заповнений “fill”

Умовні позначення:

shape – кортеж цілих чисел, що задає кількість осей та кількість включень вздовж них;

dtype – тип даних, один з типів, описаних в модулі *numpy*. Усі елементи масиву мають один і той же тип, що за замовчуванням відповідає найбільш охоплюючому типу введених значень;

order – один з варіантів ”*C*” – звичайний порядок індексів – перший номер рядка; ”*F*” – Fortran подібний – перший номер стовпчика;

subok – булеве значення, *True* – наслідується тип і підтип даних, *False* – призначається базовий підтип відповідного типу;

m – задає кількість стовпчиків у матриці *eye*;

k – служить для вибору головної діагоналі в матриці *eye*;

fill – значення, яким заповнюється матриця.

Один з оригінальних способів створення масиву зі значень функції:

```
>>> def ffn(x,y,z):  # function description
    return (2*x-y-z)

>>> nff=np.fromfunction(ffn, (3,2,2)) #matrix generat
>>> nff
array([[[ 0., -1.],
        [-1., -2.]],

       [[ 2.,  1.],
        [ 1.,  0.]],

       [[ 4.,  3.],
        [ 3.,  2.]]]).
```

Згенеровано 3-вимірний масив для значень аргументів x={0,1,2}, y={0,1}, z={0,1}.

1.4 Властивості масивів

Об'єкти *numpy.array* мають ряд властивостей і функцій для доступу до них. Перелік властивостей масивів наведено в таблиці 7.2

Таблиця 7.2 – Властивості масивів

Властивість	Значення
ndim	Повертає кількість осей (вимірів) відповідного масиву
shape	Повертає розмір масиву у вигляді кортежу цілих, що відповідає розміру масиву за кожним з вимірів. Довжина кортежу відповідає кількості вимірів – *ndim*
size	Повертає розмір масиву – повну кількість елементів – фактично добуток усіх елементів кортежу *shape*
dtype	Повертає тип елементів масиву
itemsize	Повертає розмір в байтах елемента

Приклади застосування властивостей наведено нижче.

```
>>> import numpy as np
>>> a1=np.arange(3,15)
>>> a1
array([ 3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13,
14])
>>> a1.ndim
1
>>> a1.shape
(12,)
>>> a1.size
12
>>> a1.dtype
dtype('int32')
>>> a1.itemsize
4
>>> a2=a1.reshape(3,4)
>>> a2
array([[ 3,  4,  5,  6],
       [ 7,  8,  9, 10],
       [11, 12, 13, 14]])
>>> a2.ndim
2
>>> a3=a1.reshape(2,3,2)
>>> a3
array([[[ 3,  4],
        [ 5,  6],
        [ 7,  8]],

       [[ 9, 10],
        [11, 12],
```

```
        [13, 14]]])
>>> a1.ndim, a2.ndim
(1, 2)
>>> a2.shape, a3.shape
((3, 4), (2, 3, 2))
>>> a2.dtype, a3.dtype
(dtype('int32'), dtype('int32'))
>>> a3[1,1,1]=2.3
>>> a3
array([[[ 3,  4],
        [ 5,  6],
        [ 7,  8]],
       [[ 9, 10],
        [11,  2],
        [13, 14]]])
>>> a2
array([[ 3,  4,  5,  6],
       [ 7,  8,  9, 10],
       [11,  2, 13, 14]]).
```

Як показано в коді, три змінні *a1*, *a2*, *a3* вказують на один і той же набір значень, оформлюючи його в структури різної форми. Крім того, спроба присвоїти елементу масиву з цілими значеннями *dtype('int32')* значення з плаваючою крапкою (2.3) не вдалася. Число обрізане до цілого, тип даних масиву не змінився.

Щоб змінити тип елементів масиву, необхідно створити новий об'єкт на основі існуючого, доповнивши команду явним оголошенням типу:

```
>>> b2=np.array(a3, dtype='float64')
>>> b2
array([[[  3.,   4.],
        [  5.,   6.],
```

```
        [  7.,   8.]],

       [[  9.,  10.],
        [ 11.,   2.],
        [ 13.,  14.]]])
>>> b2[0,1,0]=3.14
>>> b2
array([[[  3.  ,   4.  ],
        [  3.14,   6.  ],
        [  7.  ,   8.  ]],

       [[  9.  ,  10.  ],
        [ 11.  ,   2.  ],
         [ 13.  ,  14.  ]]]).
```

1.5 Операції над масивами

Доступ до елементів масивів можливий за іменем масиву та набором відповідних індексів, як показано у прикладі вище. Крім того, до масивів можна застосовувати операцію зрізу, подібно зрізам об’єктів типу *list* (практикум №3).

Використовуючи масив *a2* з попередніх прикладів можемо одержати:

```
>>> a2
array([[ 3,  4,  5,  6],
       [ 7,  8,  9, 10],
       [11,  2, 13, 14]])
>>> a2[1,:]
array([ 7,  8,  9, 10])
>>> a2[:,2]
array([ 5,  9, 13])
>>> a2[:1,1:2]
```

```
array([[4]])
>>> a2[:2,1:3]
array([[4, 5],
       [8, 9]]).
```

Масиви типу *numpy.array* передбачають особливі умови проведення математичних операцій як зі скалярами, так і з іншими масивами. Масив може бути використаний, як окремий операнд математичної операції. Якщо другим операндом є скаляр, то відповідна дія об'єднує цей скаляр з кожним елементом масиву, при чому результатом є масив того ж розміру:

```
>>> a=np.array([[2,3,4,5],[7,8,9,6]])
>>> a
array([[2, 3, 4, 5],
       [7, 8, 9, 6]])
>>> ab=3+a
>>> ab
array([[ 5,  6,  7,  8],
       [10, 11, 12,  9]])
>>> ac=a-2
>>> ac
array([[0, 1, 2, 3],
       [5, 6, 7, 4]])
>>> ad=a**2
>>> ad
array([[ 4,  9, 16, 25],
       [49, 64, 81, 36]]).
```

Такі ж дії можна здійснювати не тільки з елементами усього масиву, але й зрізами:

```
>>> ad[:,1:3]-=12
```

```
>>> ad
array([[ 4, -3,  4, 25],
       [49, 52, 69, 36]]).
```

Математична дія (-12) застосована до елементів стовпчиків 1 та 2 (враховуєм, що нумерація починається з 0).

Якщо обома операндами математичної операції є масиви однакового розміру, то виконується поелементна дія з утворенням нового масиву того ж розміру:

```
>>> ab
array([[ 5,  6,  7,  8],
       [10, 11, 12,  9]])
>>> ad
array([[ 4, -3,  4, 25],
       [49, 52, 69, 36]])
>>> ak=ab+ad
>>> ak
array([[ 9,  3, 11, 33],
       [59, 63, 81, 45]]).
```

Інший приклад демонструє застосування до масиву значень масиву степенів:

```
>>> ba=np.array([[3,4,5],[6,7,8],[11,12,14]])
>>> bc=np.eye(3)
>>> ba
array([[ 3,  4,  5],
       [ 6,  7,  8],
       [11, 12, 14]])
>>> bc
array([[ 1.,  0.,  0.],
```

```
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
>>> bf=ba**bc
>>> bf
array([[  3.,   1.,   1.],
       [  1.,   7.,   1.],
       [  1.,   1.,  14.]]).
```

Крім математичних дій з масивами однакового розміру, дії можна проводити з масивами, які мають сумісний розмір. Одним з випадків сумісних розмірів масивів є такий, коли обидва масиви співпадають за одним з вимірів, а інший вимір в одного з масивів дорівнює одиниці:

```
>>> cx
array([6, 7, 8])
>>> dx=ax+cx
>>> dx
array([[ 8, 10, 12],
       [12, 14, 16],
       [17, 19, 21],
       [27, 29, 31]])
>>> ex=np.array([[100],[200],[300],[400]])
>>> ex
array([[100],
       [200],
       [300],
       [400]])
>>> gx=ax+ex
>>> gx
array([[102, 103, 104],
       [206, 207, 208],
```

```
[311, 312, 313],
[421, 422, 423]]).
```

Визначення сумісності розмірів масивів та правил проведення математичних дій в таких умовах визначається поняттям трансляції (*broadcasting*) докладно описаним в офіційних посібниках з модуля *NumPy*.

1.6 Окремі корисні функції

Модуль *numpy* містить велику кількість функцій різноманітного призначення. Опис повного набору функцій виходить далеко за рамки окремого практикуму. Тому нижче наведено тільки декілька найпростіших функцій, математичний зміст яких може бути легко зрозумілим.

Функція *vdot* призначена для знаходження скалярного добутку двох векторів, заданих своїми координатами. Формат функції наступний:

```
result=np.vdot(a,b).
```

Приклад:

```
>>> v1=np.array([[3,7,4,5]])
>>> v2=np.array([[5,5,3,3]])
>>> sdd=np.vdot(v1,v2)
>>> sdd
77.
```

Функція *dot* призначена для знаходження добутку двох масивів (матриць) з розмірами придатними для проведення операції множення. Формат функції:

```
res1=dot(ax, bx)      або  dot(ax, bx[, res1])
```

Приклад:

```
>>> am=np.array([[1,2],[3,4],[5,6]])
>>> bm=np.array([[5,5,6,6],[8,9,9,7]])
>>> am
```

```
array([[1, 2],
       [3, 4],
       [5, 6]])
>>> bm
array([[5, 5, 6, 6],
       [8, 9, 9, 7]])
>>> cm=np.dot(am,bm)
>>> cm
array([[21, 23, 24, 20],
       [47, 51, 54, 46],
       [73, 79, 84, 72]]).
```

Функція *det* призначена для знаходження детермінанту (визначника) квадратної матриці. Функцію описано не в основній частині модуля *numpy*, а у розділі *linalg*, де зібрано функції лінійної алгебри. Формат виклику функції:

```
res=np.linalg.det(a).
```

Приклад:

```
>>> ad=np.array([[5,2,3],[6,5,2],[3,8,3]])
>>> ad
array([[5, 2, 3],
       [6, 5, 2],
       [3, 8, 3]])
>>> np.linalg.det(ad)
69.999999999999957.
```

2 Завдання на комп'ютерний практикум

Скласти та відлагодити програму множення прямокутних матриць за допомогою вбудованої функції модуля *numpy* та власного коду і порівняння

результатів, або іншу за завданням викладача. Зробити висновки. Для самоконтролю використовувати завдання Додатку 2.

3 Запитання для самоконтролю

1. Який тип розширення базової функціональності *Python* досягається за рахунок модуля *numpy*?

2. Чи вірні твердження:

а) масиви *numpy* можуть бути тільки двовимірними.

б) масиви *numpy* можуть містити тільки цілочисельні значення

в) масиви *numpy* є мінливими об'єктами

г) змінити тип даних масиву *numpy* неможливо

д) операція *reshape* створює новий об'єкт типу *numpy.array*

3. Що таке операція зрізу, і чи можна застосувати її до масивів *numpy*?

4. Що таке broadcasting стосовно *numpy.array*?

Література: [1] С. 607 – 614. [2] С. 187 – 190. [3] 123 – 140. [4]. [7].

Комп'ютерний практикум № 8

РОЗРОБКА ПРОГРАМИ ДЛЯ ПОБУДОВА ГРАФІКА ДОВІЛЬНОЇ ФУНКЦІЇ

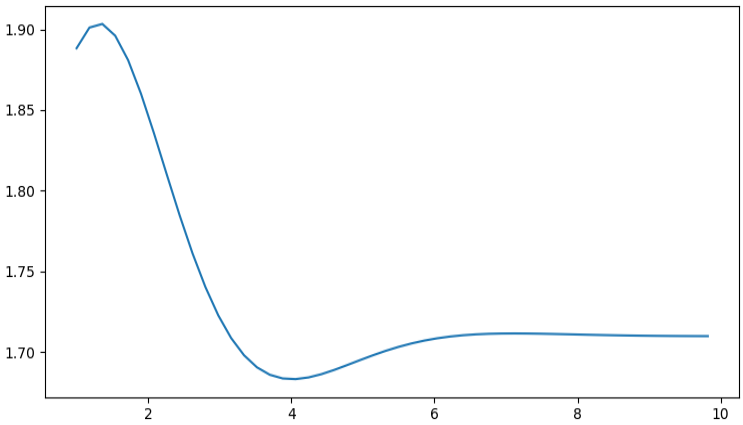
Мета роботи: оволодіння основами застосування графічних можливостей модулів *turtle* та *matplotlip.pyplot*, отримання досвіду та розробка програми для побудови графіка довільної функції на заданому інтервалі.

1 Основні теоретичні відомості

1.1 Графіки функцій та експериментальних залежностей

Графічні відображення математичних залежностей завжди відігравали суттєву роль в описі та розумінні процесів, що досліджуються. Особливо корисними з цієї точки зору є засоби комп'ютерної візуалізації математичних залежностей, експериментальних та розрахункових результатів, тощо. Порівняйте, наприклад, сприйняття виразу функції (функція є в додатку до практикуму № 2) з її графічним відображенням (рис. 8.1).

$$c = \frac{\sqrt{x} \cdot sinx}{y + e^x} + \left(\sqrt[3]{y + z}\right)$$



$y = 2, \ \ z = 3$

Рисунок 8.1 – Порівняння формули функції та її графіку

На сьогоднішній день комп'ютерна візуалізація даних та залежностей стала невід'ємною частиною не тільки інженерії (математики, фізики) а й економічних та соціальних наук, медицини, тощо. Розроблено значну кількість прикладних програм, що реалізують прийоми наукової та ділової графіки.

Середовище програмування *Python* містить декілька механізмів, які можуть бути використані для побудови графіків довільних функцій.

1.2 Побудова графіка засобами модуля *turtle*

Модуль *turtle* входить до складу базової версії *Python*, не потребує окремого встановлення та реалізує можливості ”черепашачої” графіки. Черепашача графіка – термін що застосовується до зображень, створених за допомогою передачі команд виконавцю (черепасі), які змінюють його положення, а також наявність і тип сліду, який залишає черепаха. Опис функції модуля *turtle* в Додатку 1.

Модуль *turtle* має бути імпортований, після чого команда *turtle.reset* створює графічну область з параметрами за замовчуванням.

```
>>> import turtle
>>> turtle.reset().
```

Розмір графічної області, встановленої за замовчуванням, можна перевірити та змінити:

```
>>> turtle.screensize()
400, 300)
>>> turtle.screensize(500,500)
>>> turtle.reset().
```

Створений таким чином графічний простір симетричний – точка з координатами (0, 0) знаходиться в його центрі. Межі простору і систему координат можна змінити командою *turtle.setworldcoordinates*:

```
>>> turtle.setworldcoordinates(0,0,200,100)
>>> turtle.reset().
```

Необхідно відзначити, що команди *.screensize* та *.setworldcoordinates* працюють незалежно. Перша з них встановлює розмір графічної області в пікселах екрану. Для цього використовується пара чисел: ширина та висота. Друга – створює систему координат за значеннями координат лівого нижнього та

правого верхнього кута, для чого використовується чотири числа. Таким чином може бути створена область з різними масштабами за осями координат. Команди, наведені в прикладі вище створили графічну область розміром *500 x 500* пікселів, але *500* пікселів на горизонтальній осі відповідає розміру *200* одиниць, а *500* пікселів на вертикальній осі – *100* одиниць. Ці особливості необхідно враховувати при створенні програми побудови графіку функції.

Програма побудови функції повинна включати наступні обов'язкові елементи:

– опис функції, графік якої необхідно побудувати;

– функцію, що включає інтерактивне введення меж аргументу для побудови графіка (*x0, xN*), розрахунок мінімального та максимального значення функції на інтервалі, формування меж графіка в термінах значень координат;

– створення графічного простору з заданими характеристиками;

– побудова графіка функції шляхом переміщення "черепашки" з піднятим пером в початкову точку графіка – опускання пера – покрокове переміщення "черепашки" з опущеним пером до наступних точок графіка, включаючи останню. Розглянемо складові програми детально.

Опис функції, графік якої необхідно побудувати, не викликає ускладнень, оскільки мова йде про просту математичну функцію, що повертає одне значення для одного значення аргументу.

Функція формування меж графіка має включати:

– інтерактивне введення меж аргументу: x0 та xN, перевірку та забезпечення нерівності x0 < xN;

– визначення мінімального та максимального значення функції – шляхом знаходження 15-20 значень функції в інтервалі (x0, xN);

– розширення меж графіка (в термінах координат) на 10% в кожну сторону за кожною координатою;

– повернення меж графіка у вигляді кортежу (xmin, ymin, xmax, ymax).

Створення графічного простору має включати**:**

– визначення простору командою *.screensize*;

– визначення системи координат командою *.setworldcoordinates*, передавши їй кортеж, виведений функцією формування меж;

– у випадку, якщо межі графіка включають перехід через 0 за однією чи обома координатами – відобразити осі координат.

Побудова графіка функції має включати**:**

– підняття пера;

– переміщення в точку початку графіка;

– опускання пера;

– створення циклу для поступового переходу від точки до точки з заданим кроком – не менше 50 – 100 переходів.

Переваги описаного способу полягають в повному контролі за процедурою побудови графіка. Недоліків підходу значно більше: швидкість роботи ”черепашачої” графіки дуже низька; навіть після проведеного масштабування, для виведення шкал на осях необхідно провести ряд дій.

1.3 Побудова графіка засобами модуля *matplotlib*

Модуль *matplotlib* – бібліотека призначена для візуалізації даних двовимірною 2D графікою. За способом побудови і якістю одержаних за допомогою *matplotlib* зображень, цей модуль може бути віднесений до засобів наукової графіки. Одержані результати можуть бути використані в наукових публікаціях.

Модуль *matplotlib* не входить до складу базової версії *Python*, тому має бути встановленим за допомогою інсталятора пакетів *pip*, як показано в попередніх роботах.

Для побудови графіків достатньо імпортувати частину модуля – бібліотеку *pyplot*. Загально прийнятим скороченням для бібліотеки є *plt*:

```
>>> import matplotlib.pyplot as plt.
```

Засоби модуля *matplotlib* беруть на себе завдання визначення мінімальних та максимальних значень, а також масштабування простору графіку. Більше того, визначений масштаб є динамічним – після побудови можна за допомогою миші змінити загальні розміри та пропорції, і автоматично змінити масштаб.

Для побудови необхідно передати відповідній функції (методу) масив даних, вказати особливості побудови і викликати функцію *.show()*, яка безпосередньо відповідає за відображення.

Розглянемо синтаксис декількох основних функцій побудови з бібліотеки *matplotlib.pyplot:*

*".plot(*args, **kwargs)* – побудова ліній та/або маркерів в системі осей, які залежать від аргументів **args*.".

Приклади застосування:

– *plot(x,y)* – побудова графіка за масивами координат x та y з використанням типу лінії та кольору за замовчуванням.

– *plot(x, y,'bo')* – побудова графіка за тими ж масивами з використанням маркерів у вигляді синіх кругів.

– *plot(y)* – побудова графіка за масивом y, де в якості координати x використовуються значення індексів масиву y.

Засоби форматування:

‘- ‘ суцільна лінія

‘-- ‘ пунктирна лінія

‘ -. ‘ штрих-пунктирна лінія

‘ : ‘ лінія з крапок

‘. ‘ маркери у вигляді крапок

‘, ‘маркери у вигляді пікселів

‘o ‘маркери у вигляді кругів

‘v ‘маркери у вигляді трикутників вершиною вниз

‘^ ‘маркери у вигляді трикутників вершиною вгору

‘< ‘маркери у вигляді трикутників вершиною вліво

‘> ‘маркери у вигляді трикутників вершиною вправо

‘s ‘ маркери у вигляді квадратів

‘p ‘ маркери у вигляді п’ятикутників

‘* ‘маркери у вигляді зірок

‘h ‘маркери у вигляді шестикутників 1.

‘H ‘ маркери у вигляді шестикутників 2

‘ + ‘маркери у вигляді знаків +

‘x ‘маркери у вигляді діагональних хрестиків

Засоби шифрування кольорів

' b '	blue	синій
' g '	green	зелений
' r '	red	червоний
' c '	cyan	блакитний
'm'	magenta	пурпурний
'y'	yellow	жовтий
'k'	black	чорний
'w'	white	білий

Кольори можуть бути задані і іншими способами.

В якості **kwargs** можуть бути задані деякі додаткові прийоми форматування графічного зображення:

plt.plot([1,2,3], [1,2,3], 'go-', label='line 1', linewidth=2) – побудова графіку суцільною лінією, що зв’язує маркери, зеленим кольором, позначення *line 1*, товщина 2 пікселя.

plt.plot([1,2,3], [1,4,9], 'rs', label='line 2') – побудова графіку без лінії, червоні квадрати, позначення ***line 2.***

plt.scatter(x, y, s=None, c=None, marker=None, cmap=None, norm=None, vmin=None, vmax=None, alpha=None, linewidths=None, verts=None, edgecolors=None, hold=None, data=None, **kwargs) – побудова розсипного графіка (набору маркерів).

Принциповою відмінністю від використання *.plot* без ліній є можливість змінювати розмір маркера.

Наприклад команда:

```
>>>plt.scatter([1,2,3,4,5], [1,2,3,2.5,2],
[35,22,58,40,35])
 >>>plt.show().
```

виводить графік, показаний на рис. 8.2 – символи мають розмір пропорційний елементам третього масиву, вказаного в виклику функції.

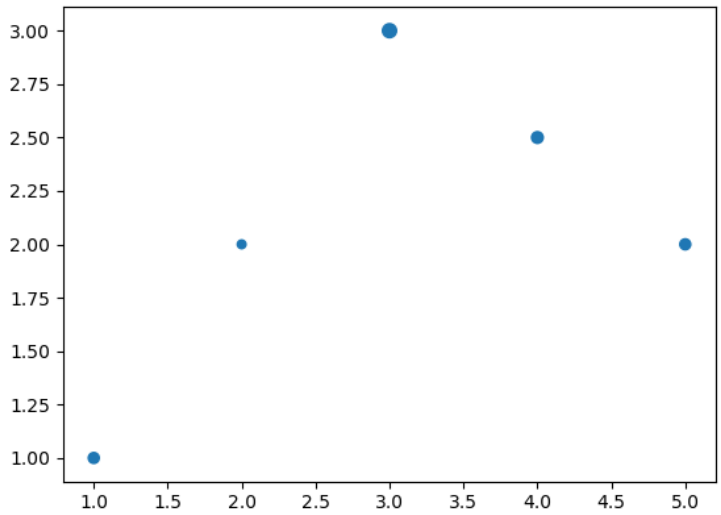


Рисунок 8.2 – Графік одержаний з використанням функції *scatter*

.legend() – виведення легенди в поле графіка

.show() – відтворення графіка.

Для побудови графіка з використанням засобів модуля *matplotlib* програма має включати лише:

– опис функції, графік якої необхідно побудувати – співпадає з аналогічним кроком для побудови графіка засобами модуля *turtle*;

– діалог для введення інтервалу побудови функції: інтерактивне введення меж аргументу: *x0 та xN*, перевірку та забезпечення нерівності $x0 < xN$;

– формування масиву значень аргументу *xar*;

– формування масиву значень функції, що відповідають значенням аргументу;

– виклик функцій побудови – методів модуля matplotlib.pyplot.

2 Завдання на комп'ютерний практикум

Скласти та відлагодити програми побудови графіків 2-х функцій згідно індивідуальних завдань на роботу № 2 на заданих викладачем інтервалах. Одну з функцій побудувати з використанням модуля *turtle*, іншу – модуля *matplotlib*.

3 Запитання для самоконтролю

1. Чи потрібен крок масштабування при побудові графіка засобами модуля *turtle*? Для чого?
2. Чи потрібен крок масштабування при побудові графіка засобами модуля *matplotlib*? Для чого?
3. Чи можливо вивести на графік і символи і лінії? Як це залежить від модуля, який використовується для побудови?

Література: [1] – [8]. [3] С. 141 – 151.

Комп'ютерний практикум № 9

РОЗВ'ЯЗОК АЛГЕБРАЇЧНИХ РІВНЯНЬ МЕТОДОМ ДОТИЧНИХ (НЬЮТОНА)

Мета роботи: набуття практичного досвіду розв'язку алгебраїчних рівнянь методом дотичних (Ньютона), а також отримання практичного використання числових методів для розв'язання прикладних задач.

1 Основні теоретичні відомості

1.1 Розв'язання нелінійного рівняння методом дотичних

Якщо алгебраїчне рівняння достатньо складне, то його корені досить рідко вдається знайти аналітично, а значить точно. Крім того, у ряді випадків рівняння може містити коефіцієнти, значення яких відомі лише наближено, а значить сама постановка задачі про знаходження точних коренів втрачає зміст. Саме тому, важливе значення мають способи наближеного знаходження коренів алгебраїчних рівнянь та оцінка їх точності.

Нехай задано рівняння:

$$f(x) = 0, \tag{9.1}$$

де функція $f(x)$ визначена та неперервна на деякому інтервалі $a < x < b$.

В окремих випадках можуть бути необхідними існування та неперервність першої $f'(x)$, а інколи навіть другої $f''(x)$ похідних. Всяке значення ξ, що перетворює значення функції в нуль: $f(\xi) = 0$, називається коренем рівняння (9.1) або нулем функції.

Для спрощення, будемо вважати, що рівняння (9.1) має лише ізольовані корені, або іншими словами, що для кожного кореня рівняння існує окіл, що не містить інших коренів цього рівняння. Наближене знаходження ізольованих дійсних коренів рівняння (9.1) зазвичай складається з двох етапів:

1. Виокремлення коренів – встановлення досить малих інтервалів (a, b) на яких міститься один і тільки один корінь рівняння (9.1).

2. Уточнення наближених коренів – доведення їх до заданого рівня точності.

Для відокремлення коренів часто користуються наслідками теореми: Якщо неперервна функція $f(x)$ має значення різних знаків на кінцях відрізка (α, β), $f(\alpha) \cdot f(\beta) < 0$, то такий інтервал містить щонайменше один корінь рівняння (9.1), такий, що $f(\xi)$. Корінь ξ виявиться єдиним коренем рівняння, якщо похідна $f'(x)$ існує і зберігає свій знак на усьому інтервалі (α, β).

Процес відокремлення коренів, у зв’язку зі сказаним вище, доцільно починати з встановлення знаків функції $f(x)$ в граничних точках області її

існування. Потім необхідно знайти значення функції у окремих проміжних точках інтервалу. Результатом таких дій являється набір інтервалів (a_i, b_i) кожен з яких містить корінь рівняння (9.1).

В загальному випадку, область визначення функції може бути і $(-\infty, +\infty)$, а проміжні точки необхідно вибирати напівінтуїтивно, керуючись загальним виглядом рівняння. Однак, для більшості реальних технічних задач, початковий інтервал, чи наближене місце знаходження кореня чи коренів відоме, і визначається особливостями фізичних процесів та обмеженнями технічних можливостей. Якщо відомий початковий інтервал, і він є досить малим, для умов конкретної задачі, можна безпосередньо переходити до уточнення коренів. Якщо початковий інтервал досить великий, доцільно виконати декілька повторів методу поділу інтервалу навпіл, для більш точної локалізації коренів.

Одним з найпростіших в реалізації методів розв'язання алгебраїчних рівнянь – є метод дотичних, який називають ще і методом Ньютона.

Нехай корінь ξ рівняння $f(x) = 0$ знаходиться на проміжку (a, b), причому $f'(x)$ та $f''(x)$ неперервні та зберігають свої знаки для усіх x, що належать цьому проміжку. Знайшовши деяке n-е наближення значення кореня $x_n \approx \xi$, $(a \leq x_n \leq b)$, його можна уточнити за методом Ньютона.

Нехай:

$$\xi = x_n + h_n, \tag{9.2}$$

де h_n вважаємо малою величиною. Звідси, застосувавши формулу Тейлора, одержимо:

$$f(\xi) = 0 = f(x_n + h_n) \approx f(x_n) + h_n f'(x_n). \tag{9.2.1}$$

А значить:

$$\mathrm{h_n} = -\frac{\mathrm{f(x_n)}}{\mathrm{f'(x_n)}}. \tag{9.2.2}$$

Якщо внести відповідну поправку у формулу (9.2), знайдемо вираз для наступного наближення кореня:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \qquad (n = 0,1,2, \dots) \ . \tag{9.3}$$

Геометрично метод Ньютона еквівалентний заміні невеликої дуги кривої $y = f(x)$ дотичною, проведеною у точці поточного наближення розв'язку (рис. 9.1).

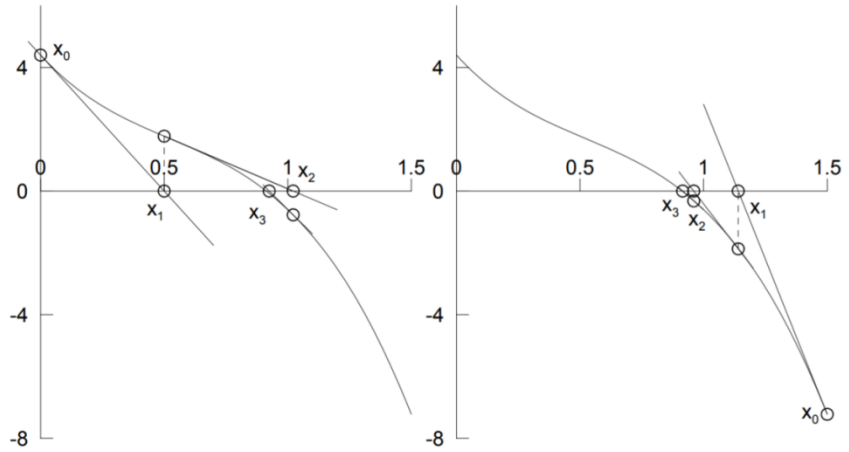


Рисунок 9.1 – Графічний вигляд ітерацій метода Ньютона залежно від положення початкової точки.

Таким чином, використовуючи метод Ньютона, можна знайти корінь рівняння з наперед заданою точністю. Для оцінки похибки n-го наближення x_n можна користуватись загальною формулою:

$$|\xi - x_n| \leq \frac{|f(x_n)|}{m_1}, \tag{9.4}$$

де m_1 – найменше значення $|f'(x)|$ на відрізку (a, b).

Виходячи з того, що в області, близькій до кореня, для великої кількості рівнянь, значення похідної $f'(x)$ практично не змінюється, то за критерій зупинки пошуку можна використовувати нерівність:

$$\left|\frac{f(x_n)}{f'(x_n)}\right| < \varepsilon \,, \tag{9.5}$$

де ε – задана точність обчислень.

1.2 Розробка програми

Програма, що реалізує метод Ньютона повинна містити:

– опис функції,

– процедуру інтерактивного введення наближення,

– обчислення похідної,

– знаходження наближених коренів,

– оцінку точності,

– спосіб оцінки того, збігається обчислювальний процес чи розбігається,

– виведення результатів обчислень.

Розкриємо зміст частин більш детально.

Опис функції. Алгебраїчне рівняння в програмі може задаватися за допомогою простого опису функції. Наприклад:

```
def fn1(x):
    return(x**4+2*x**3-x-1).
```

Процедура інтерактивного введення наближення. Початкове наближення необхідно одержувати від користувача у діалоговому режимі за допомогою стандартних операторів вводу-виводу.

Обчислення похідної. Для універсальності програми, обчислення похідної доцільно проводити чисельно, використовуючи формулу:

$$f'(x_i) \approx \frac{f(x_i+\delta)-f(x_i-\delta)}{2\delta} \,, \tag{9.6}$$

де δ – достатньо мале значення більше від нуля.

У програмі це може бути реалізовано у вигляді додаткової функції користувача. Оскільки середовище *Python* дозволяє передавати ім'я функції в якості аргумента, опис функції для чисельного одержання значення похідної може мати вигляд:

```
def dfdx(fn, x0, dlt=2e-2 ):
    return((fn(x0+dlt)-fn(x0-dlt))/(2*dlt).
```

Таким чином, в функцію чисельного диференціювання передаються: назва функції, значення аргументу в поточній точці та, за потреби, значення δ. Якщо останнє не вказано, використовується значення за замовчуванням: 0,02.

Знаходження наближених коренів проводиться за формулою (9.3) наведеною в теоретичній частині.

Оцінка точності в програмі обчислюється за формулою (9.5) і виступає одним із критеріїв завершення процесу. Очевидно, що в такому випадку, ітераційний процес пошуку кореня має бути організований у вигляді циклу з умовою.

Для оцінки того, збігається обчислювальний процес чи ні, необхідно оцінювати абсолютне значення функції на послідовних ітераціях. Якщо абсолютне значення функції на кожній наступній ітерації менше попереднього, то можна вважати, що метод збігається. Якщо ж на деякій ітерації відбувається збільшення значення функції, то необхідно проаналізувати ще одну ітерацію.

Вивід результатів обчислень організується як просте виведення останнього одержаного перед зупинкою циклу значення наближеного кореня.

За необхідності наведення графічних результатів розв'язання нелінійного рівняння необхідно обрати зручний метод графічної побудови, оцінити можливий інтервал значень аргументу та функції, за потреби – провести масштабування області екрану по вертикалі та горизонталі, вивести графік рівняння, точки, що знаходяться на послідовних ітераціях, та відрізки дотичних декількох початкових ітерацій.

2 Завдання на комп'ютерний практикум

Розробити, скласти та відлагодити програму розв'язання нелінійного рівняння методом дотичних з виведенням чисельного значення кореня, кількості проведених ітерацій, побудови демонстраційних графіків з використанням бібліотеки *matplotlib.pyplot* або *turtle*.

3 Запитання для самоконтролю

1. Як зміниться швидкість (необхідна кількість ітерацій) знаходження кореня з заданою точністю, якщо замінити чисельне (наближене) значення похідної на точне?
2. В яких випадках відсутня збіжність ітераційного процесу?
3. Які інші наближені методи розв'язання нелінійних рівнянь Ви знаєте?

Література: [1] – [5].[3] С. 25 – 33. [6] С. 127 – 153. [8].

Комп'ютерний практикум № 10

РОЗРОБКА ПРОГРАМИ ДЛЯ АПРОКСИМАЦІЇ (НАБЛИЖЕННЯ) ТАБЛИЧНИХ ДАНИХ ПОЛІНОМОМ ЗА МЕТОДОМ НАЙМЕНШИХ КВАДРАТІВ

Мета роботи: закріплення теоретичних відомостей та оволодіння практичним досвідом використання методу найменших квадратів для апроксимації експериментальних даних.

1 Основні теоретичні відомості

1.1 Основи методу найменших квадратів

Задачі апроксимації експериментальних даних функціями заданого вигляду досить часто зустрічаються в інженерній та науковій практиці. На відміну від інтерполяційних формул (розглядаються в програмі 2-го семестру), коли шукають

поліном, значення якого у вузлових точках точно дорівнюють табличним значенням, апроксимація передбачає наближення таблиці даних функцією заданого вигляду, в окремому випадку – поліномом заданої степені. В цьому випадку значення апроксимуючої функції не обов'язково дорівнює табличним значенням (графік не обов'язково проходить через усі табличні точки), але описує весь їх набір найкращим способом. Застосування апроксимації доцільне у цілому ряді випадків, в першу чергу, коли вигляд функції відомий або табличні точки знайдено (виміряно) з деякою похибкою.

Процедура побудови функції заданого вигляду за табличними даними аналогічна (у випадку функції однієї змінної) процедурі побудови графіка, наприклад, прямої за набором точок. На рисунку 10.1 показано побудову графіка за набором точок. Суцільною лінією показана лінійна залежність, а штрихованою – квадратична.

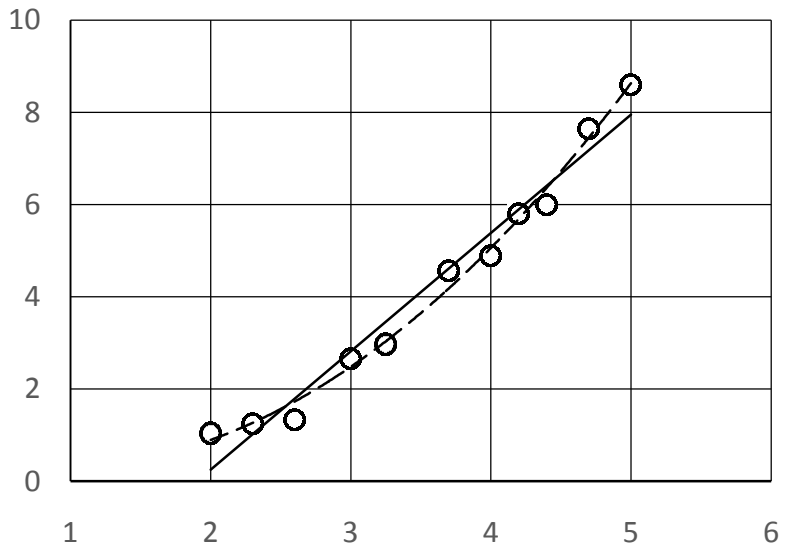


Рисунок 10.1 – Лінійна (суцільна лінія) та квадратична (штрихова лінія) залежності одержані за методом найменших квадратів згідно таблиці даних (значки O)

Підхід методу найменших квадратів полягає в формуванні функції сумарного розбігу таблиці та апроксимуючої функції за усіма точками таблиці та пошуку мінімуму такої функції. Розглянемо процедуру методу докладніше.

Нехай задано таблицю даних у вигляді Kпар значень незалежної змінної (x) та функції $f(x)$, як схематично показано нижче:

x	x_1	x_2	x_3	$\cdots$	x_K
$f(x)$	$f(x_1)$	$f(x_2)$	$f(x_3)$	$\cdots$	$f(x_K)$

Для простоти обмежимо вигляд функції поліномом n-ї степені, загальний вигляд якого:

$$h(x) = A_0 + A_1 x + A_2 x^2 + \cdots + A_n x^n \tag{10.1}$$

Завданням апроксимації є знаходження таких коефіцієнтів рівняння (10.1), які забезпечать найменше відхилення $h(x_i)$ від $f(x_i)$ для усіх K точок таблиці.

Одним зі способів розв'язання задачі є пряме формулювання системи лінійних рівнянь – для кожної точки таблиці $(x_i, \quad f(x_i))$ записується рівняння:

$$A_0 + A_1 x_i + A_2 x_i^2 + \cdots + A_n x_i^n = f(x_i), \qquad i = 1,2,\cdots,K \tag{10.2}$$

Таким чином, для K точок таблиці буде сформульовано K лінійних рівнянь з $(n+1)$ невідомими A_j. Така система має єдиний розв'язок (може бути розв'язана методами лінійної алгебри) тільки в тому випадку, якщо кількість рівнянь дорівнює кількості невідомих коефіцієнтів $K = n + 1$. У випадку, якщо $K < n + 1$ система не має єдиного розв'язку, а значить коефіцієнти рівняння (10.1) не можуть бути знайдені.

Якщо $K > n + 1$, що відповідає більшості реальних ситуацій, система рівнянь є перевизначеною, і може не мати розв'язку взагалі. Для одержання наближеного розв'язку систему (10.2) перепишемо у вигляді:

$$\begin{cases} A_0 + A_1x_1 + A_2x_1^2 + \cdots + A_nx_1^n - f(x_1) = q_i \\ A_0 + A_1x_2 + A_2x_2^2 + \cdots + A_nx_2^n - f(x_2) = q_2 \\ \cdots \\ A_0 + A_1x_K + A_2x_K^2 + \cdots + A_nx_K^n - f(x_K) = q_K \end{cases}$$

Права сторона системи рівнянь (q_i) є набором функцій, що називаються розбігами. Сформуємо функцію сумарного розбігу:

$$Q(A) = \sum_{i=1}^{K} q_i^2,$$

мінімум якої відповідає такому набору коефіцієнтів A_j, який забезпечує найкраще наближення полінома до таблиці значень, а значить, розв'язок задачі апроксимації.

Функція сумарного розбігу при постійній таблиці даних залежить виключно від набору коефіцієнтів $A_0, A_1, \cdots, A_n$ причому завдяки своєму вигляду вона має мінімум, який може бути достатньо просто знайдений.

Необхідною умовою існування мінімуму функції Q у деякій точці (для деякого набору коефіцієнтів A_j) є рівність нулю усіх частинних похідних. Таким чином ми одержимо нову систему рівнянь:

$$\begin{cases} \dfrac{\partial Q}{\partial A_0} = 2 \cdot \sum_{i=1}^{K} \left(A_0 + A_1x_i + A_2x_i^2 + \cdots + A_nx_i^n - y_i\right) = 0 \\ \dfrac{\partial Q}{\partial A_1} = 2 \cdot \sum_{i=1}^{K} \left(A_0 + A_1x_i + A_2x_i^2 + \cdots + A_nx_i^n - y_i\right) \cdot x_i = 0 \\ \cdots \\ \dfrac{\partial Q}{\partial A_n} = 2 \cdot \sum_{i=1}^{K} \left(A_0 + A_1x_i + A_2x_i^2 + \cdots + A_nx_i^n - y_i\right) \cdot x_i^n = 0 \end{cases} \quad (10.3)$$

Остання система є системою $n + 1$ лінійних рівнянь з $n + 1$ невідомими, що має єдиний розв'язок – набір значень $A_0, A_1, \cdots, A_n$ - коефіцієнтів, що забезпечують найкраще наближення табличних даних. Кожне рівняння системи

включає постійний вираз в дужках – фактично розбіг на і-й точці, та змінну величину, що є фактично виразом частинної похідної по відповідному Aj від виразу в дужках.

Після спрощення виразів і приведення до зручного вигляду, система (10.3) може бути записана у вигляді:

$$\begin{cases} A_0K + A_1\sum_{i=1}^{K} x_i + A_2\sum_{i=1}^{K} x_i^2 + \cdots + A_n\sum_{i=1}^{K} x_i^n = \sum_{i=1}^{K} y_i \\ A_0\sum_{i=1}^{K} x_i + A_1\sum_{i=1}^{K} x_i^2 + A_2\sum_{i=1}^{K} x_i^3 + \cdots + A_n\sum_{i=1}^{K} x_i^{n+1} = \sum_{i=1}^{K} y_i x_i \\ \cdots \\ A_0\sum_{i=1}^{K} x_i^n + A_1\sum_{i=1}^{K} x_i^{n+1} + A_2\sum_{i=1}^{K} x_i^{n+2} + \cdots + A_n\sum_{i=1}^{K} x_i^{2n} = \sum_{i=1}^{K} y_i x_i^n \end{cases} \quad (10.4)$$

Одержана система є нормально визначеною системою лінійних рівнянь, яку можна розв’язати будь-яким доступним методом.

Розглянемо приклад. Функцію задано таблицею даних:

x	2.5	2.8	3.5	4.5	5.5	5.8	6.5
$y(x)$	1.2	0.5	0.2	1.3	4.2	5.8	9.5

За допомогою методу найменших квадратів знайти коефіцієнти поліномів:

1. $h(x) = A_0 + A_1x$

2. $g(x) = B_0 + B_1x + B_2x^2$

Використовуючи одержану раніше систему лінійних рівнянь (10.4), запишемо систему ідентифікації для функції $h(x)$ з урахуванням таблиці даних, що містить сім точок.

$$\begin{cases} A_0 \cdot 7 + A_1 \cdot \sum_{i=1}^{7} x_i = \sum_{i=1}^{7} y_i \\ A_0 \cdot \sum_{i=1}^{7} x_i + A_1 \cdot \sum_{i=1}^{7} x_i^2 = \sum_{i=1}^{7} y_i x_i \end{cases}$$

Знайдемо значення сум:

$$\sum_{i=1}^{7} x_i = 31.1 \qquad \sum_{i=1}^{7} y_i = 22.7$$

$$\sum_{i=1}^{7} x_i^2 = 152.73 \qquad \sum_{i=1}^{7} y_i x_i = 129.44$$

Запишемо систему лінійних рівнянь з числовими коефіцієнтами

$$\begin{cases} 7 \cdot A_0 + 31.1 \cdot A_1 = 22.7 \\ 31.1 \cdot A_0 + 152.73 \cdot A_1 = 129.44 \end{cases}$$

Розв'язання системи лінійних рівнянь[4] дає результат: $A_0 = -5.48$, $A_1 = 1.96$.

Аналогічно запишемо систему лінійних рівнянь для квадратичної апроксимуючої функції:

$$\begin{cases} B_0 \cdot 7 + B_1 \cdot \sum_{i=1}^{7} x_i + B_2 \cdot \sum_{i=1}^{7} x_i^2 = \sum_{i=1}^{7} y_i \\ B_0 \cdot \sum_{i=1}^{7} x_i + B_1 \cdot \sum_{i=1}^{7} x_i^2 + B_2 \cdot \sum_{i=1}^{7} x_i^3 = \sum_{i=1}^{7} y_i x_i \\ B_0 \cdot \sum_{i=1}^{7} x_i^2 + B_1 \cdot \sum_{i=1}^{7} x_i^3 + B_2 \cdot \sum_{i=1}^{7} x_i^4 = \sum_{i=1}^{7} y_i x_i^2 \end{cases}$$

[4] Методи чисельного розв'язання систем лінійних рівнянь розглянуто в програмі 2-го кредитного модуля. Для розв'язання систем рівнянь можна застосовувати знання одержані в розділі «Лінійна алгебра» або застосовувати автоматизовані способи доступних систем електронних таблиць, або середовища програмування

Або у форматі чисельних коефіцієнтів:

$$\begin{cases} 7 \cdot B_0 + 31.1 \cdot B_1 + 152.73 \cdot B_2 = 22.7 \\ 31.1 \cdot B_0 + 152.73 \cdot B_1 + 807.69 \cdot B_2 = 129.44 \\ 152.73 \cdot B_0 + 807.69 \cdot B_1 + 4492.43 \cdot B_2 = 763.73 \end{cases}$$

Результат розв'язку другої системи: $B_0 = 12.43,\ B_1 = -7.07,\ B_2 = 1.02$.

Залежно від конкретного випадку (мета побудови залежності, рівень оцінки статистичної істинності тощо) точність проведеної апроксимації може оцінюватись за різними підходами та алгоритмами. Найпростішим способом є розрахунок середньоквадратичного відхилення розрахункових значень від табличних за формулою:

$$S = \sqrt{\frac{1}{K}\sum_{i=1}^{K}\left(f(x_i) - g(x_i)\right)^2}, \qquad (10.5)$$

де $f(x_i)$ та $g(x_i)$ табличне значення та значення апроксимуючої функції, відповідно у точці x_i.

Для розрахованих апроксимуючих функцій обчислені середньоквадратичні відхилення складуть: $S_h = 3.87$, $S_g = 0{,}09$, що свідчить, що апроксимація квадратичним поліномом $g(x)$ значно точніша, ніж лінійною залежністю $h(x)$.

1.2 Розробка програми

Програма, що реалізує метод найменших квадратів має містити:

1. Блок формування масивів вихідних даних,

2. Процедуру формування розширеної матриці системи лінійних рівнянь згідно степені апроксимуючого полінома,

3. Розв'язання системи лінійних рівнянь,

4. Обчислення середньоквадратичного відхилення,

5. Вивід чисельних результатів,

6. Вивід графіків табличної функції (окремі символи) та апроксимуючої функції.

Розкриємо зміст частин більш детально.

1. Формування масивів вихідних даних – має передбачити введення в діалоговому режимі кількості точок, значень незалежної змінної та функції в кожній з точок, степінь апроксимуючого поліному. При розробці та тестуванні програми вказані параметри доцільно задати значеннями в тексті програми.

2. Формування матриці системи лінійних рівнянь – досить відповідальний етап, під час виконання якого необхідно враховувати вигляд апроксимаційної функції (зокрема поліноміальний), а також особливості мови програмування та розмір вхідної таблиці даних. Для мов програмування, які не містять вбудованих функцій степені, або такі функції громіздкі, доцільно створювати додатковий масив, в якому поступово накопичувати степінь значень незалежного параметру x.

Враховуючи, що для поліноміальної апроксимуючої функції степінь для значень x під знаком суми змінюється від 0 до $2n$, доцільно створити окремий масив розмірності $2n + 1$, в якому зібрати значення сум, необхідних для формування матриці системи.

Нескладно помітити, що індекс елемента в такому масиві відповідає сумі індексів в матриці системи (при базі індексації = 0).

При застосуванні функцій обробки масивів модуля *numpy* процедуру можна суттєво скоротити:

Будемо вважати, що `xa` та `ya` є об'єктами класу *numpy.ndarray*, в яких розміщено табличні дані незалежної змінної та функції відповідно. Тоді кількість точок таблиці може бути знайдена як `len(xa)`, сума елементів `xa` записується як `sum(xa)`, сума квадратів елементів `xa` як `sum(xa**2)`. Подібним чином сума добутків значень незалежної змінної на значення функції запишеться, як `sum(xa*ya)`.

Повністю фрагмент програми для створення масиву сум, може мати наступний вигляд:

```
sums_x=[len(xa)]# ініціалізація масиву
for i in range(1,2*n+1):
    sums_x.append(sum(xa**i)),
```

а цикл формування матриці системи можна записати як:

```
amatr=np.empty((n,n), dtype=float)  # ініціалізація порожньої матриці
for i in range(n+1):
    for j in range(n+1):
        amatr[i,j]=sums_x[i+j].
```

Подібним чином створюється масив елементів правої частини матриці – стовпчик вільних членів.

3. Розв'язання створеної системи лінійних рівнянь може бути проведено або процедурою за методом Гауса (необхідна окрема програма) або з використанням можливостей модуля *numpy*, зокрема функції *numpy.linalg.solve*, виклик якої має формат:

```
a=np.linalg.solve(amatr, rmatr)
```

де ***amatr*** – матриця, формування якої описано вище,

rmatr – стовпчик вільних членів,

a – масив коефіцієнтів апроксимуючого полінома.

4. Для виконання подальших кроків зручно створити функцію, яка матиме в якості вхідної інформації масив коефіцієнтів полінома та окреме значення незалежної змінної і буде повертати значення апроксимуючої функції. Вигляд функція може мати наступний:

```
def mypoly(a, x):
    vvl=a[0,0]
    for i in range (1, len(a)):
        vvl+=a[i,0]*x**i
    return vvl.
```

Застосування такої функції спрощує одержання значення апроксимуючого полінома в довільній точці, а значить забезпечує можливість виконання кроків 4-6.

Для побудови графіка необхідно, додатково до існуючої таблиці даних, створити таблицю значень апроксимуючої функції з кроком, значно меншим за середній крок початкової таблиці. Кількість точок такої таблиці має складати не менше 50.

2 Завдання на компютерний практикум

Розробити, скласти та відлагодити програму апроксимації таблиці даних поліномом степені від 1 до 4 за методом найменших квадратів з побудовою графіка.

3 Запитання для самоконтролю

1. Як пов’язані степінь апроксимуючого полінома та розмір системи лінійних рівнянь?
2. Яка найвища степінь значення незалежної змінної під знаком суми використовується для побудови кубічного полінома?
3. Які методи розв’язання систем лінійних рівнянь Ви знаєте, і які з них зручно застосовувати при розв’язанні системи в рамках методу найменших квадратів?

Література: [1] – [5]. [6] С. 225 – 234. [7]. [8].

ДОДАТОК 1

ОПИС ОСНОВНИХ ФУНКЦІЙ МОДУЛЯ TURTLE

Модуль *turtle* дозволяє застосовувати графічні примітиви обома шляхами: об'єктно-орієнтованим та процедурно-орієнтованим. Основою для роботи *turtle* є графічна підкладка *tkinter*, тому модуль *tkinter* має бути в складі встановленого пакету *Python*.

Д.1.1 Методи руху маркера *turtle*

Д.1.1.1 Методи переміщення та створення зображень

– *turtle.forward(distance)* або *turtle.fd(distance)*: переміщення маркера – «черепахи» в поточному напрямку на відстань *distance*, задану в одиницях розміру робочої області екрану. Залежно від стану пера, маркер залишає, чи не залишає слід.

– *turtle.back(distance),* або *turtle.bk(distance)*, або *turtle.backward(distance)*: переміщення маркера в напрямку, протилежному до поточного на відстань *distance*, задану в одиницях розміру робочої області екрану. Залежно від стану пера, маркер залишає чи не залишає слід.

– *turtle.right(angle),* або *turtle.rt(angle),turtle.left(angle),* або *turtle.lt(angle)*: поворот маркера на кут *angle* вправо, або вліво (за замовчуванням кут задається в градусах, але може бути змінений функціями *degrees()* та *radians()*). Орієнтування кутів залежить від налагоджування простору функцією *mode()*.

– *turtle.goto(x, y=None),* або *turtle.setpos(x, y=None),* або *turtle.setposition(x, y=None)*: переміщує маркер в позицію, задану абсолютними координатами. Залежно від стану пера, маркер залишає, чи не залишає слід. Операція не змінює кут орієнтування маркера.

Позиція задається або двома значеннями, що відповідають координатам x та y, або одним значенням, яке має бути екземпляром класу *turtle.Vec2D(x, y)* – кортеж, який повертається, на приклад, функцією *turtle.pos()*.

– *turtle.setx(x), turtle.sety(y):* встановлює значення першої, або другої координат маркера, залишаючи іншу з координат без змін. Залежно від стану пера, маркер залишає чи не залишає слід. Операція не змінює кут орієнтування маркера.

– *turtle.setheading(to_angle),* або *turtle.seth(to_angle)*: встановлює кутову орієнтацію маркера в напрямок *to_angle*, заданий в градусах. Напрям прямої з кутом «0» і позитивний напрям відліку кутів залежать від режиму, встановленого функцією *turtle.mode()*.

– *turtle.home()*: повертає маркер в нульову позицію (0,0) і орієнтує його в напрямку кута 0 (що залежить від режиму, встановленого функцією turtle.*mode*()). Залежно від стану пера, маркер залишає, чи не залишає слід.

Маркер повертається до положення (0, 0), навіть в тому випадку, якщо ця точка не входить в поточний графічний простір, заданий функцією *turtle.setworldcoordinates()*.

– *turtle.circle(radius, extent=None, steps=None)*: створює зображення кола заданого радіусу – *radius*, або правильного багатокутника, вписаного в коло того ж радіусу – якщо вказано значення параметру *steps*. Якщо задано параметр *extent*, то відображається не повне коло (багатокутник), а його частина, що відповідає значенню *extent* в градусах: 90 – чверть кола, 180 – пів кола, тощо. Значення радіусу (*radius*) та кута дуги (*extent*) можуть бути як позитивними, так і негативними. Позитивний радіус – побудова кола в напрямі позитивного відліку кутів, негативний – негативного відліку кутів. Позитивне значення – побудова в напрямку маркера, негативний – в зворотному напрямку. Місце розташування побудованого об’єкта залежить від поточного стану маркера (координати, напрямок) та від режиму, встановленого функцією *turtle.mode()*.

– *turtle.dot(size=None,*color)*: створює зображення круглої “точки” розміром *size* в поточному положенні маркера. Значення – ціле число ≥1, якщо не задане явно – використовується більше з двох значень – подвійна ширина пера чи ширина пера + 4. Колір точки визначається параметрами сумісними з функцією *turtle.color()*.

– *turtle.stamp( )*: створює відбиток поточної форми маркера в його поточній позиції і повертає ціле число, яке є ідентифікатором (*id*) цього відбитку. Передбачає можливість подальшого видалення.

– *turtle.clearstamp(stampid)*: видаляє відбиток за його ідентифікатором (*id*) – *stampid*.

– *turtle.clearstamps(n=None)*: видаляє усі або вказану кількість відбитків. Якщо параметр *n* не задано, то видаляються усі відбитки, прикріплені до рисунка. Якщо *n>0,* то видаляються *n* – відбитків починаючи з початку, якщо *n<0* то видаляються *n* – відбитків починаючи з кінця.

– *turtle.undo()*: відміняє останню дію, виконану в програмному режимі *turtle*. Послідовне виконання функції дозволяє відмінити дію декількох останніх операцій. Кількість операцій, які можна відмінити залежить від розміру програмного буфера функції *turtle.undo()*.

– *turtle.speed(speed=None)*: встановлює швидкість переміщення маркера, і відповідно швидкість створення зображення. Виклик функції без аргументу повертає поточну швидкість маркера у відносних одиницях. Якщо вказане значення більше 10 або менше 0,5, то встановлюється швидкість «0», що не передбачає анімації, а зображення створюються з максимально можливою швидкістю.

Швидкість може бути задана назвою:

"*fastest*": 0 – найвища, без анімації

"*fast*": 10 – найвища з анімацією

"*normal*": 6 – середня

"*slow*": 3 – повільна

"*slowest*": 1 – найнижча.

Д.1.1.2 Методи стану маркера

– *turtle.position( ) або turtle.pos( )*: повертає координати поточного положення маркера *(x, y)*. Результат повертається у вигляді кортежу, який має формат *Vec2D*, який може бути аргументом функції *turtle.goto()*.

– *turtle.towards(x, y=None)*: повертає кут між поточним положенням і напрямом маркера та напрямом на задану точку. Координати точки задаються або двома скалярними значеннями або одним кортежем, що містить два значення. Значення кута залежить від режиму, встановленого функцією *turtle.mode()*.

– *turtle.xcor( ), turtle.ycor( )*: повертає x, або y координату поточного положення маркера.

– *turtle.heading()*: повертає поточний напрям маркера – кут, значення якого залежить від режиму, встановленого функцією *turtle.mode()*.

– *turtle.distance(x, y=None)*: повертає відстань від поточного положення маркера до заданої точки. Координати точки задаються або двома скалярними значеннями, або одним кортежем, що містить два значення.

Д.1.1.3 Методи призначення кутів

– *turtle.degrees (fullcircle=360.0):* встановлює розмір одиниць вимірювання кутів. Значення за замовчуванням – стандартні градуси. Можуть бути встановлені інші одиниці – вказується кількість одиниць на кут рівний повному колу.

– *turtle.radians( ):* встановлює режим вимірювання кутів у радіанах.

Д.1.2 Методи управління пером

Д.1.2.1 Методи стану пера

– *turtle.pendown( )*, або *turtle.pd ( )*, або *turtle.down( )*: переводить перо об’єкта *turtle* в режим створення зображення. Переміщення маркера буде викликати появу ліній, чи інших примітивів.

– *turtle.penup( )*, або *turtle.pu ( )*, або *turtle.up( )*: переводить перо об'єкта *turtle* в режим «підняте». Переміщення маркера не буде викликати появу зображення. Застосовується для перенесення маркера у місце початку нового зображення.

– *turtle.pensize(width=None)*, або *turtle.width (width=None)*: встановлює ширину пера рівну *width* і повертає її, як значення. Якщо метод застосовано без аргумента – повертає поточне значення ширини пера.

– *turtle.pen (pen=None, **pendict)*: повертає та/або дозволяє замінити один або декілька параметрів пера. Параметром *pen* методу передається словник (*dictionary*[5]) з раніше збереженим набором параметрів, що дозволяє повернутись до відповідних характеристик пера. Параметр ***pendict* включає одну чи декілька пар, ключове слово – значення, які необхідно змінити в стані пера. Застосовуються наступні параметри та значення:

Таблиця Д1.1 – Параметри, що відображають характеристику пера

Параметр	**Значення**	**Призначення**
1	2	3
shown	True / False	Маркер *turtle* видимий / невидимий
pendown	True / False	Перо опущене (режим зображення) / підняте
pencolor	color-string / color-tuple	Задає колір ліній створення зображення у вигляді назви або кортежу
fillcolor	color-string / color-tuple	Задає колір заливки у вигляді назви або кортежу
pensize	Positive number	Задає розмір (ширину) пера
speed	Number 0..10	Задає швидкість створення зображення
resizemode	auto або user або noresize	Задає режим зміни розмірів маркера
stretchfactor	Positive number, positive number	Задає відношення вертикальних та горизонтальних одиниць вимірювання
outline	positive number	Задає розмір обрамлення маркера
tilt	Number	Задає кут нахилу маркера без зміни напрямку рисування

[5] Застосування класу *dictionary* не розглядається в рамках даного циклу лабораторних робіт.

– *turtle.isdown ()*: повертає значення *True*, якщо поточний стан пера відповідає режиму створення зображення. В іншому випадку функція повертає значення *False*.

Д.1.2.2 Методи стану кольору

– *turtle.pencolor (*args)*: повертає або встановлює колір пера. Можливі 3 варіанти формату цього методу:

– *turtle.pencolor( )*: повертає поточне значення кольору в одному з форматів – як назву кольору або як кортеж компонентів кольору. Одержане значення може бути використане для встановлення кольору іншою командою *pencolor*.

– *turtle.pencolor(colorstring)*: встановлює значення кольору пера, використовуючи назву кольору – *colorstring*, одну з передвстановлених назв кольорів, яка передається методу у вигляді рядка.

– *turtle.pencolor(r, g, b)* – встановлює значення кольору пера, використовуючи компоненти *r – red, g – green, b – blue*. Чисельні значення компонентів кольору залежать від режиму кольору, встановленого методом.

– *turtle.fillcolor (*args)*: повертає, або встановлює колір заповнення. Можуть використовуватись усі варіанти формату, описані для методу *pencolor()*.

– *turtle.color (*args)*: повертає, або встановлює колір пера та колір заповнення. Для кожного з аргументів (перо чи заповнення) може використовуватись специфікація кольорів, описана для методу *pencolor()*. Метод може викликатись з специфікацією одного кольору, або специфікаціями двох кольорів:

– *turtle.color(colorstring)* – встановлюється однаковий колір для пера та заповнення;

– *turtle.color(colorstring1, colorstring2)* – встановлюється колір *colorstring1* для пера та *colorstring2* для заповнення.

Д.1.2.3 Інші методи управління зображенням

– *turtle.reset ( )*: видаляє зображення з екрану, центрує розміщення графічної області, переміщає маркер в початкову точку і надає усім змінним значення за замовчуванням.

– *turtle.clear( )*: видаляє зображення з екрану. Розміщення та орієнтування маркера, а також значення змінних залишається без змін.

– *turtle.write(arg, move=False, align=”left”, font=(“Arial”, 8, “normal”))*: виводить текст *(arg)* починаючи з поточної позиції маркера. Використовуються наступні аргументи при виклику методу:

arg – рядкова константа (задана безпосередньо, або через ідентифікатор);

move – може приймати значення *True* або *False*. Параметр відповідає за те переміщується, чи не переміщується маркер при виводі тексту. Якщо необхідно виводити частини тексту різними викликами методу *write,* то потрібно надавати параметру значення *True*;

align – параметр вирівнювання тексту, може приймати значення *”left”, ”center”* або*”right”*;

font – специфікація шрифту, що включає кортеж з трьома компонентами: назва шрифту, розмір шрифту, тип шрифту.

Література: [5].

ДОДАТОК 2

ЗАДАЧІ ДЛЯ САМОСТІЙНОЇ РОБОТИ ТА КОНТРОЛЮ

До комп'ютерного практикуму № 2

Формули для програмування та значення для випадку значень змінних: *x=1, y=2, z=3*

Таблиця Д2.1 – Формули та значення змінних

Формула	Значення
1	2
$a = \left(\sqrt{x-1} + \frac{1}{y-1}\right) \cdot e^{2cosz}$	0.13807
$b = \frac{x}{y} - \frac{y}{2z} ln(z + 3e^{yx})$	-0.57518
$c = \frac{\sqrt{x} \cdot sinx}{y + e^{x}} + \left(\sqrt[3]{y+z}\right)$	1.88832
$d = \frac{e^{sin^3 x} + \ln(arctgy)}{sin\, x + \cos z} \cdot \sqrt[3]{x+z}$	-20.48158
$f = -\pi + \left(\frac{x+y}{1-xy}\right) \cdot z \cdot \sqrt[5]{x^2+y^2}$	-15.559
$g = \frac{\sqrt{\lvert x-1\rvert} - \sqrt{\lvert y\rvert}}{1 + \frac{x^2}{2} + \frac{z^2}{4}} \cdot log_3 xz - sin\frac{\pi}{x}$	-0.37712
$h = \frac{3 + e^{2x}}{1 + x^2\lvert y - tgz\rvert} + z^{e^x}$	3.45735
$k = 1 + \lvert y - x\rvert + \frac{(y-x)^2}{2} + \frac{(x-z)^2}{3} - \sqrt[3]{y^2+z^2}$	1.481999
$l = (1+y)\frac{x + \frac{y}{z^2 - lnz}}{e^2 + \frac{z}{x^2+y^2}}$	0.470564

Продовження таблиці 12.1

1	2
$m = \frac{1 + cos^3(x-y)}{\frac{x^2}{2} + sin^2 z} \cdot \sqrt[3]{z^2 - y^2}$	3.80772
$n = \frac{1 + sin^2\left(x - \sqrt{y}\right)}{2 + \left\lvert \frac{2z}{1+x^2y^2} - 7 \right\rvert} \cdot \sqrt[3]{xyz}$	0.270700
$p = log_3 \left\lvert \left(y - \sqrt{\lvert x \rvert}\right) \cdot \left(x - \frac{y}{z + \frac{e^x}{4}}\right) \right\rvert$	-0.713862
$q = (2 + x) \frac{3 + \frac{x+y^2}{y-z^2}}{\left\lvert y^z + \frac{sin^2 x}{x+z} \right\rvert} + \sqrt[3]{xyz}$	2.65571
$r = \frac{1 + sin^2(x+z) + cos^3(z-y)}{lnx + \left\lvert x - \frac{2x}{1+xyz} \right\rvert}$	2.42267

До комп'ютерного практикуму № 3

1. Скласти таблицю 10 цілочисельних значень, записаних в двійковій, восьмеричній, шіснадцятиричній та десятковій системах числення: значення з різних інтервалів (0..10), (2**2 .. 2**5), (2**6-2**10); не менше 4 чисел <0.

2. З деякого набору значень з плаваючою крапкою програма згенерувала таблицю:

x	*x.as_integer_ratio()*
0.1	*(3602879701896397, 36028797018963968)*
0.125	*(1, 8)*
0.08333333333333333	*(6004799503160661, 72057594037927936)*
0.0625	*(1, 16)*
0.1875	*(3, 16)*
0.2	*(3602879701896397, 18014398509481984)*
0.09375	*(3, 32)*

0.08571428571428572 (3088182601625483, 36028797018963968).

Пояснити одержані результати. Підібрати значення x, які можуть підтвердити висновок.

3. Скласти та виконати програму для реалізації п. 9 контрольних запитань. Результати вивести в форматі:

n1=ml[:] [1, ..., 10] 10 elements

4. Скласти та виконати програму для демонстрації наступних властивостей списків:

– *append;*

– *insert* (має бути спроба вставки в кінець списку, в середину списку, за межі існуючих індексів списку);

– *pop;*

– *remove* (має бути спроба видалення елементу з початку списку, з середини списку, із-за межі існуючих індексів списку).

До комп'ютерного практикуму № 4

Використовуючи оператори циклу для відомих N, a та b реалізувати наступні вирази (операції степені та факторіалу не застосовувати):

$$c = \left(1 + \frac{1}{1^2}\right) \cdot \left(1 + \frac{1}{2^2}\right) \times \cdots \times \left(1 + \frac{1}{N^2}\right)$$

$$d = \frac{1}{a} + \frac{2}{a^2} + \cdots + \frac{N}{a^N}$$

$$f = (1 - a) + (2 + a^2) + (3 - a^3) + \cdots + (N + (-1^n) \cdot a^n)$$

$$g = \frac{a}{1!} + \frac{2a}{2!} + \cdots + \frac{N \cdot a}{N!}$$

$$h = (a - b) + (a^2 - b^2) + (a^3 - b^3) + \cdots + (a^N - b^N)$$

$$k = (N + a) \cdot (N - 1 - a^2) \cdot (N - 2 + a^3) \times \cdots \times (1 + (-1^N) \cdot a^N)$$

$$j = (1+2)\cdot(1+2+3)\cdot(1+2+3+4)\times\cdots\times(1+2+\cdots+N)$$

$$l = \left(1+\frac{a}{1!}\right)\cdot\left(1+\frac{a^2}{2!}\right)\cdot\left(1+\frac{a^3}{3!}\right)\times\cdots\times\left(1+\frac{a^N}{N!}\right)$$

$$m = \left(1+\frac{a}{1!}\right)-\left(1+\frac{a^2}{2!}\right)+\left(1+\frac{a^3}{3!}\right)-\cdots+(-1)^{N-1}\cdot\left(1+\frac{a^N}{N!}\right)$$

$$p = \left(a+\frac{1}{1}\right)\cdot\left(a^2+\frac{1}{1+2}\right)\cdot\left(a^3+\frac{1}{1+2+3}\right)\times\cdots$$
$$\times\left(a^N+\frac{1}{1+2+3+\cdots+N}\right)$$

$$q = \left(a+\frac{1}{1}\right)+\left(a^2+\frac{1+2}{2}\right)+\left(a^3+\frac{1+2+3}{3}\right)+\cdots$$
$$+\left(a^N+\frac{1+2+3+\cdots+N}{N}\right)$$

До комп'ютерного практикуму № 7

1. Розробити програму для одержання добутку двох матриць: $\mathbf{C} = \mathbf{A}\cdot\mathbf{B}$

а) $\mathbf{A} = \begin{pmatrix} 2 & 3 & 2 & 5 \\ 3 & 1 & 2 & 3 \\ 5 & 2 & 1 & 1 \end{pmatrix}$ $\mathbf{B} = \begin{pmatrix} 3 & 2 \\ 4 & 2 \\ 1 & 7 \\ 5 & 3 \end{pmatrix}$ $\mathbf{C} = \begin{pmatrix} 45 & 39 \\ 30 & 31 \\ 29 & 24 \end{pmatrix}$

б) $A = \begin{pmatrix} 8 & 4 & 2 \\ 3 & 1 & 7 \\ 2 & 5 & 4 \\ 4 & 4 & 2 \end{pmatrix}$ $B = \begin{pmatrix} 3 & 2 & 4 & 6 \\ 4 & 2 & 2 & 1 \\ 1 & 7 & 2 & 6 \end{pmatrix}$

$$C = \begin{pmatrix} 42 & 38 & 44 & 64 \\ 20 & 57 & 28 & 61 \\ 30 & 42 & 26 & 41 \\ 30 & 30 & 28 & 40 \end{pmatrix}$$

2. Розробити рекурсивну функцію для знаходження значення детермінанту за допомогою алгебраїчних доповнень

Матриця $\mathbf{A} = \begin{bmatrix} 10 & 3 & 2 & 3 & 3 \\ 2 & 11 & 8 & 7 & 0 \\ 6 & 10 & 7 & 8 & 3 \\ 8 & 12 & 6 & 2 & 4 \\ 8 & 6 & 4 & 3 & 11 \end{bmatrix}$ Детермінант $|\mathbf{A}| = 4228$

ДОДАТОК 3

ЗАДАЧІ ДЛЯ РОЗРАХУНКОВОЇ РОБОТИ

1. Задано масив цілих чисел M[20]. Вивести індекс його останнього парного елемента.

2. Задано масив дійсних чисел P[20]. Вивести кількість його додатних елементів та суму їх індексів.

3. Задано масив цілих чисел K[20]. Перетворити його за правилом: парні елементи розділити націло на цілу частину середнього арифметичного від'ємних елементів.

4. Задано масив дійсних чисел K[20]. Розмістити його елементи за зростанням, не відкриваючи нового масиву.

5. Задано масив дійсних чисел P[20]. Вивести добуток його додатних елементів та їх кількість.

6. Задано масив дійсних чисел K[20]. Сформувати масив B[20], елементами якого будуть різниці між сумою усіх елементів попереднього масиву та кожним з них.

7. Задано масив цілих чисел M[20]. Підрахувати та вивести кількість парних, непарних, додатних та від'ємних чисел.

8. Задано масив дійсних чисел M[20]. Пронормувати масив – кожен елемент масиву виразити як різницю його самого та середнього арифметичного усіх елементів розділену на середнє арифметичне усіх елементів.

9. Задано масив дійсних чисел P[20]. Вивести суму його найбільшого та найменшого елементів та суму їх індексів.

10. Задано масив цілих чисел K[20]. Вивести суму його елементів, що менші за їх індекси.

11. Задано матрицю K[4,4]. Знайти та вивести максимальні елементи кожного стовпчика.

12. Задано матрицю T[7,8]. Вивести кількість додатних, від'ємних та рівних нулю елементів.

13. Задано матрицю A[6,6]. Вивести добуток від'ємних елементів та суму додатних елементів.

14. Задано масив цілих чисел T[18]. Створити два нові масиви T1 та T2, включаючи до одного парні, а до другого непарні числа.

15. Задано дві послідовності цілих чисел A1[8] та A2[7]. Побудувати об'єднану послідовність A[15] з розміщенням елементів за зростанням.

16. Знайти та вивести усі елементи цілочисельної матриці M[7,7], які дорівнюють сумі їх індексів (i+j).

17. Задано матрицю M[5,7]. Сформувати нову матрицю L[7,5], елементами якої є частки від ділення елементів транспонованої вихідної матриці на їх суму.

18. Задано матрицю K[6,5]. Вивести її найбільший та найменший елементи та їх індекси.

19. Заповнити матрицю M[12,15] числами, що рівні добутку індексів. Елементи кратні 3 замінити на 43, кратні 7 на 47, кратні і 3 і 7 на 11. (рекомендовано створити функцію кратності).

20. Задано матрицю цілих чисел M[6,6]. Сформувати нову матрицю B[6,6], елементами якого є частки від ділення парних елементів вихідної матриці на суму додатних і непарних елементів на суму від'ємних.

21. Задано матрицю P[4,8]. Вивести індекси її найбільшого та найменшого елементів.

22. Задано два масиви A[4,5] та B[5,7]. Сформувати одновимірний масив AB[?], що складається з від'ємних елементів першого, а потім другого масивів.

23. Задано матрицю P[6,4]. Вивести суму її найбільшого та найменшого елементів.

24. У масиві P[30] усі елементи являють собою 0, 1 або 2 у випадковому порядку. Розмістити їх так, щоб спочатку були нулі, потім – одиниці і нарешті двійки. Додаткового масиву не відкривати.

25. Задано матрицю A[7,4]. Знайти та вивести максимальні елементи кожного рядка.

26. Задано матрицю M[7,4]. Вивести суму її від'ємних елементів та їх кількість.

27. Задано матрицю M[6,8]. Обнулити усі її від'ємні елементи.

28. Задано натуральне число N (N<1000). Знайти та вивести масив A[?], який включає усі прості числа менші заданого.

29. Задано 2 натуральних числа K1 і K2. Знайти та вивести найбільший спільний дільник, реалізувавши алгоритм Евкліда.

30. Задано масив цілих чисел P[35]. Вивести елементи масиву, які кратні 3 та некратні 5.

31. Задано масив координат точок X[20,2]. Серед його елементів знайти точки, які належать колу заданого радіуса, побудованого з центра координат.

32. В графічному режимі побудувати на екрані 11 кіл, кожне наступне з яких менше попереднього на 1/3 і зсунуте по горизонталі на 2/3 радіуса. Використовувати цикл.

33. В графічному режимі побудувати на екрані 10 рівносторонніх трикутників, кожний наступний з яких має сторону в 1,3 рази більшу попереднього, а центри лежать на одній горизонтальній прямій на відстані 20 пікселів один від одного. Використовувати цикл.

34. В графічному режимі побудувати ромб з координатами вершин: (300, 50), (500, 200), (300, 350), (100, 200). Побудувати п'ять інших ромбів, центри яких співпадають з центром заданого, а довжина сторони кожного з них складає 0,8 довжини сторони попереднього. Використовувати цикл.

35. В графічному режимі побудувати 10 квадратів, кожен наступний з яких менше попереднього на 1/5 і повернутий відносно попереднього на 10º. Використовувати цикл.

ДОДАТОК 4

ПРАВИЛА ОФОРМЛЕННЯ РОЗРАХУНКОВОЇ РОБОТИ

Індивідуальне завдання на розрахункову роботу формується за допомогою програмного селектора і включає 7 задач зі списку, наведеного в додатку 3, або інших задач, подібних за типом та складністю. Кожне завдання включає одну з задач (28-31) та одну з задач (32-35).

Виконана розрахункова робота повинна мати вигляд зшитої записки, яка включає:

– титульний аркуш (зразок наведено у кінці додатку);

– текст програми мовою *Python*, що реалізує розв'язок відповідної задачі (текст оформлюється в рукописному вигляді, кожна задача оформлюється з нового аркуша, приклад оформлення наведено в кінці додатку);

– список посилань на використану літературу, оформлений згідно вимог чинного стандарту;

– додаток у вигляді магнітного чи оптичного носія з текстами програм.

Додаток може бути замінений посиланням на електронні версії програм завантажені через систему «електронний кампус "КПІ ім. Ігоря Сікорського".

ДОДАТОК 5

ТИТУЛЬНИЙ АРКУШ

Національний технічний університет України

"Київський політехнічний інститут імені Ігоря Сікорського"

Кафедра високотемпературних матеріалів та порошкової металургії

РОЗРАХУНКОВА РОБОТА

з дисципліни – "Інформатика, обчислювальна техніка, програмування та числові методи "

Кредитний модуль 1 - "Інформатика, обчислювальна техніка та програмування"

Виконав(ла):

Студ гр. ФН-71

Коваленко А.В.

Керівник

Доц. Михайлюк М.М.

Захищено з оцінкою_____

Київ – 20ХХ

ДОДАТОК 6

ПРИКЛАД ОФОРМЛЕННЯ ПРОГРАМИ

Задача 1. Задано масив цілих чисел F[15]. Знайти та вивести мінімальне та максимальне значення масиву, а також їх різницю.

Програма

```
#Аналіз масиву - пошук мінімального та максимального числа
#та їх різниці
n=15 #кількість елементів
f=[7,3,8,9,12,35,71,23,37,44,2,18,77,43,51]# масив для аналізу
#---Аналіз стандартними засобами програмування---
fmin=f[0]
fmax=f[0]
for i in range(15):
    if fmin>f[i]:
        fmin=f[i]
    elif fmax<f[i]:
        fmax=f[i]
print('common approach')
print('min = ', fmin, ' max = ', fmax)
print('difference (max-min) = ', fmax-fmin)
#---Аналіз засобами аналізу Python---
pmin=min(f)
pmax=max(f)
print('Python approach')
print('min = ', pmin, ' max = ', pmax)
print('differnce (max-min) = ', pmax-pmin)
```

Результат роботи програми:

```
common approach
min =  2  max =  77
differnce (max-min) =  75
Python approach
min =  2  max =  77
differnce (max-min) =  75
```

ПЕРЕЛІК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ

1. Лутц Марк. Изучаем Python [Текст] / Марк Лутц ; [пер. с англ.].-– СПб. : Символ-Плюс, 2011. – 1280 с.

2. Maruch Stef. Python for Dummies [Text] / Stef Maruch, Maruch Aahz.– John Wiley & Sons, 2006.– 434 p.

3. Introduction to Computation and Programming Using Python [Текст] / John V. Guttag. The MIT Press. Cambridge, Massachusetts; London, England. – 2013. – 298 p.

4. Python 3.7.3 documentation : the Python Tutorial [Електронний ресурс]. – Режим доступу : (https://docs.python.org/3/tutorial/index.html). – Назва з екрану. 29.03.2018. Дата опублікування: 28.04.2019.

5. Documentation : The Python Standard Library 24 : Program Frameworks, 24.1. turtle. – Turtle graphics [Електронний ресурс]. Режим доступу : https://docs.python.org/3/library/turtle.html. – Назва з екрану. – Дата опублікування: 28.04.19.

6. Hoffman Joe D. Numerical methods for engineers and scientists [Текст] / Joe D. Hoffman. Basel: Marcel Dekker. Inc., New York. – 2001. – 824 p.

7. NumPy basics [Електронний ресурс] – Режим доступу : https://docs.scipy.org/doc/numpy/user/basics.html. – Назва з екрану. –Дата опублікування 31.01.2019.

8. Matplotlib User's Guide 3.1.0 [Електронний ресурс] – Режим доступу : (https://matplotlib.org/users/index.html). – Назва з екрану – Дата опублікування: 18.05.2019.